Received: 7 March 2017 | Revised: 24 May 2017 | Accepted: 26 May 2017

DOI: 10.1002/rob.21732

REGULAR ARTICLE

WILEY

Autonomous aerial navigation using monocular visual-inertial fusion

Yi Lin* | Fei Gao* | Tong Qin* | Wenliang Gao* | Tianbo Liu | William Wu | Zhenfei Yang | Shaojie Shen

Hong Kong University of Science and Technology

Correspondence
Yi Lin, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong.
Email: ylinax@connect.ust.hk

*These authors contributed equally to this work.

Abstract

Autonomous micro aerial vehicles (MAVs) have cost and mobility benefits, making them ideal robotic platforms for applications including aerial photography, surveillance, and search and rescue. As the platform scales down, MAVs become more capable of operating in confined environments, but it also introduces significant size and payload constraints. A monocular visual-inertial navigation system (VINS), consisting only of an inertial measurement unit (IMU) and a camera, becomes the most suitable sensor suite in this case, thanks to its light weight and small footprint. In fact, it is the minimum sensor suite allowing autonomous flight with sufficient environmental awareness. In this paper, we show that it is possible to achieve reliable online autonomous navigation using monocular VINS. Our system is built on a customized quadrotor testbed equipped with a fisheye camera, a low-cost IMU, and heterogeneous onboard computing resources. The backbone of our system is a highly accurate optimization-based monocular visual-inertial state estimator with online initialization and self-extrinsic calibration. An onboard GPU-based monocular dense mapping module that conditions on the estimated pose provides wide-angle situational awareness. Finally, an online trajectory planner that operates directly on the incrementally built three-dimensional map guarantees safe navigation through cluttered environments. Extensive experimental results are provided to validate individual system modules as well as the overall performance in both indoor and outdoor environments.

KEYWORDS

aerial robotics, mapping, planning, position estimation

1 | INTRODUCTION

Micro aerial vehicles (MAVs) are ideal robotic platforms for a wide range of applications in indoor and outdoor environments thanks to their low-cost, small size, and superior mobility. Due to the fast dynamics of an MAV and the possibility of non-line-of-sight operations, it is essential that the aerial robot is capable of autonomous navigation.

Reliable state estimation, environment mapping, and obstacle avoidance are the foremost important modules for autonomous navigation. Global Navigation Satellite Systems (GNSS) provide a straightforward solution for state estimation, enabling the successful commercialization of a large number of outdoor applications for autonomous micro aerial vehicles. However, GNSS do not provide any obstacle information. They are also unavailable for operations in confined, cluttered, or indoor environments that demand the mobility advantages of MAVs.

Small platforms are needed for operations in confined environments. However, smaller platforms usually come with tight size, weight, and power (SWaP) constraints, limiting their ability to carry active but heavy sensors such as radars or LiDARs. The dimensions of a small platform also limit its ability to carry stereo or multicamera systems due to insufficient baseline length. As the platform becomes smaller, a monocular visual-inertial system (VINS), consisting of only a low-cost inertial measurement unit (IMU) and a camera, becomes the only viable sensor setup due to its light weight and small footprint. In fact, the monocular VINS, which is widely available on consumer electronic devices, is the minimum sensor suite allowing autonomous flights with sufficient environmental awareness.

Despite the excellent SWaP characteristics, there are significant challenges when using a monocular VINS for autonomous navigation. Distance is not directly observable with a single camera, and to obtain metric measurements from a monocular VINS, reliable tracking of features in the environment, together with motions with sufficient acceleration and angular velocities, are required. Measurements from the IMU and the camera are fused together to obtain metric estimates of the state. This is a fragile and highly nonlinear process

that requires wide field-of-view (FOV) cameras for feature tracking, accurate camera-IMU extrinsic calibration, and good estimator initialization. The use of dense 3D maps is necessary to ensure detection of obstacles that are not covered by sparse features. To this end, highly accurate pose estimation, as well as powerful computing platforms, are required for the generation of dense depth maps through per-pixel multiview triangulation, outlier removal, and depth map fusion. To close the perception-action loop, we also need trajectory planning and feedback control modules that operate on the reconstructed dense environment representation to guarantee safety.

In this paper, we present a complete system solution to address all the aforementioned challenges. The system-level contribution is the proof of the feasibility of using the minimum monocular VINS and fully onboard processing to achieve safe navigation through unknown cluttered environments. The decisions on each system module, from hardware to software, are results from careful engineering considerations and tradeoffs. We choose to use a monocular fisheye camera with IMU-triggered hardware synchronization to provide wide-angle observations with precise time stamping. We use a set of powerful heterogeneous onboard computing platforms for parallelizing the computationally demanding per-pixel image operations.

At the level of software modules, our contributions are embedded into the choice of suitable real-time algorithms that are either developed specifically for this work, or are improved from our previous works. The backbone of our algorithm package is a high-accuracy, tightly coupled, optimization-based monocular visual-inertial state estimator with fisheye camera support, online initialization, and camera-IMU extrinsic calibration. A plane-sweeping-based multiview depth estimation module, in combination with semiglobal depth smoothing, provides real-time dense depth images for obstacle detection. The depth images are fused into a global 3D map using a probabilistic truncated signed distance function (TSDF) fusion. Finally, a gradient-based trajectory planning module, which directly operates on the reconstructed 3D map, provides collision-free trajectories through cluttered 3D environments. The perception-action loop is closed by executing the desired trajectory using a standard multirotor controller. To the best of our knowledge, we are the first to achieve such navigation capability using this minimum sensor suite. This work points out a direction toward autonomy for very small (less than 10 cm in diameter) aerial vehicles.

The rest of this paper is divided into eight parts. System-level related works are briefly discussed in Section 2. Section 3 gives an overview of our system in both hardware and software architectures. Section 4 discusses how we use the monocular fisheye camera. State estimation, dense 3D mapping, and trajectory planning modules are presented in Section 5, Section 6, and Section 7, respectively. Online experimental results (Section 8) verify the performance of individual modules, as well as the integrated system. We show autonomous navigation in both indoor and outdoor environments. Section 9 concludes this work and points out possible directions for future improvements, where we aim to extend our framework toward systems with higher speed, lower power consumption, smaller size, and lower cost.

2 | RELATED WORK

There is a rich body of scholarly work on autonomous flight in GNSS-denied indoor and outdoor environments. The collection becomes even more extensive when we consider individual modules that contribute to the overall system. In this section, we only review the most relevant system-level works that use low-cost visual sensing modalities, and leave the discussion on individual modules to their corresponding sections.

Stereo vision-based methods have been widely studied for aerial navigation. In Ref. 1, three onboard computing units are used, with FPGA for real-time stereo-matching, Gumstix for real-time embedded processing, and an Intel Core2Duo for visual odometry, mapping, and planning. By using a probabilistic voxel map provided by Octomap library,[2] a path planner is designed to provide an obstacle-free path to a designated waypoint. A loosely coupled visual-inertial extended Kalman filter (EKF) is used for state estimation. However, due to the spatial baseline requirement for stereo depth estimation, it is hard to miniaturize the platform. Similarly, Ref. 3 gives another vision-based mapping and planning framework where its mapping module runs offboard. A loop closure module is also implemented using offboard processing. However, besides a couple of forward-looking stereo cameras, another down-looking camera is used for optical flow-based state estimation. In our work, however, we show how only a single camera is sufficient to achieve all navigational functionalities.

Monocular vision-based approaches using various sensor placement and computing infrastructures have been proposed. Combining the SVO[4] and REMODE,[5] a low weight and low cost flight platform is designed[6] to fly autonomously and perform dense 3D reconstruction simultaneously in real-time. The platform is equipped with a downward camera for both state estimation and mapping, which is insufficient for obstacle detection in cluttered environments. In addition, Ref. 6 offloads the mapping module to a ground station, which introduces high delays to a point that is unsuitable for online navigation. By only using a smartphone that contains a sufficient sensor including an IMU and a monocular camera, Ref. 7 achieves autonomous flight in GPS-denied environments. However, no mapping or obstacle avoidance modules are developed in this work. Most similar to our system configuration is Ref. 8, where a forward-looking monocular visual-inertial system is used for closed-loop autonomous navigation. However, the depth estimation module is still implemented offboard using the semidense approach.[9]

In Ref. 10, the authors present a two-layer fusion framework with multiple sensors aiming to develop a robust autonomous navigation system in challenging environments such as a shipboard environment. With a redundant sensor suite, Ref. 10 mainly focuses on providing a safety path and presents convincing robust navigation experiments in an extreme environment. However, with the different objective, we contemplate about building an agile flight platform with minimum sensing.

Visual navigation can also be realized on fixed wing aerial vehicles. Based on an IMU and a planar laser, Ref. 11 proposes a novel extension of the Gaussian particle filter to solve the localization problem. By demonstrating challenging indoor environment experiments, the

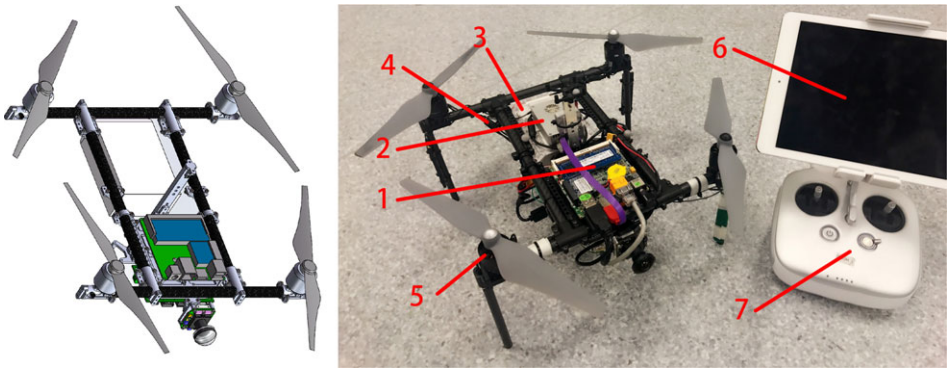


FIGURE 1 A synthetic and closeup of our aerial robot platform: (1) Perception Core; (2) DJI Lightbridge 2; (3) DJI Phantom 4 Pro battery; (4) DJI Takyon Z425-M ESC; (5) DJI E310 motors; (6) iPad Pro; (7) radio controller

authors give a well thought out on high-speed flight in known indoor environments.

Another direction of research involves the use of learning-based techniques for end-to-end generation of navigation actions. Representative works include Refs. 12 and 13. Despite being a promising direction, learning-based methods have not yet reached the level of maturity for safe flight in complex unknown environments.

3 | SYSTEM OVERVIEW

We now present the hardware design and the software architectures of our quadrotor experimental testbed. We address requirements in sensor selection, sensor synchronization, distribution of computing power, monitoring, debugging tools, experimental interfaces, and modular system designs. We will explain the rationale behind each of our system design decisions along with the descriptions of system modules.

3.1 | Hardware architecture

We aim to design a modular, compact, durable, and easy-to-construct aerial robot testbed. Our airborne system consists of two main modules: the *Perception Core* and the *Quadrotor Frame*. The perception core integrates onboard computers and sensors, and the quadrotor frame consists of the mechanical construction, motors, propellers, and the battery.

3.1.1 | Perception core

The perception core is a detachable module that houses all computing and sensing modalities. It enables independent testing of the state estimation and mapping modules and can be used with other mechanical frames.

We use two computers to form a powerful heterogeneous computing platform (Figure 2) . One is a mini computer that is powered by a dual-core Intel i7-5500U processor running up to 3.00 GHz.* It is equipped with 8 GB memory and 256 GB SSD, and consumes around 15 W of power. Another onboard computer is a NVIDIA TX1, which is

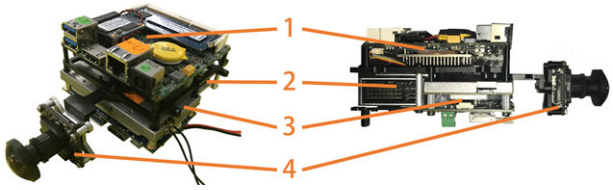


FIGURE 2 Two views of the Perception Core. (1) Mini i7 computer; (2) DJI A3 flight controller with integrated IMU; (3) NVIDIA TX1; (4) monocular camera with a fisheye lens

TABLE 1 Weight and power consumption of components in the perception core

Component	Weight (g)	Power (W)
Mini i7 computer	127	15(avg)
NVIDIA TX1	120	15(avg)/60(max)
mvBlueFOX-MLC200w	24	2.5(max)
DJI A3	66	8(max)

assembled by a core† on a carrier board.‡ The 256 NVIDIA CUDA GPU cores on the TX1 make it particularly suitable for parallel computing of depth images and TSDF fusion. The TX1 is equipped with a quad-core ARM Cortex-A57 processor and 4 GB memory which consumes approximately 15 W of power. More details about each component in the Perception Core can be found in Table 1. The two onboard computers are connected using an Ethernet cable. Communication between two computers is done by utilizing the ROS infrastructure.§

Our onboard computing resources cover most of the widely used development platforms: x86, ARM, and GPU, making it highly flexible for the development of advanced algorithms. It significantly reduces the development cycle since researchers do not need to spend excessive time on code-level optimization. However, we stress that our onboard resources, if compared by numbers, are just on-par with state-of-the-art cellphone processors such as the Qualcomm SnapDragon 835. This implies that algorithms developed using the perception core can be transferred onto low-power mobile computing platforms after sufficient code-level optimizations.

* https://ark.intel.com/products/85214/Intel-Core-i7-5500U-Processor-4M-Cache-up-to-3_00-GHz

† http://www.nvidia.com/object/jetson-tx1-dev-kit.html

‡ http://www.connecttech.com/sub/products/ASG003.asp

§ http://wiki.ros.org/

An onboard camera and IMU are also included in the perception core. We use a forward-looking MatrixVision mvBlueFOX-MLC200w* global shutter camera with 752×480 resolution. It is outfitted with a 220-degree ultra-wide-angle fisheye lens. A DJI A3 flight controller† is used both as the IMU and attitude stabilization controller. We disable the GPS and magnetometer functionalities in A3 due to their lack of reliability in cluttered indoor environments.

Accurate time stamps are required for high-performance visual-inertial fusion. In our system, this is achieved via hardware synchronization, where individual image capture is triggered by synchronization signals from the DJI A3 flight controller. More specifically, we utilize the triggering model in the Hardware Real-Time Controller‡ of the camera. The DJI Onboard SDK§ provides an interface for generating a bundle of the pulse signal and the corresponding IMU measurements. Note that the IMU runs at a higher frequency than the camera, and an IMU measurement will be tagged if it corresponds to an image. Only one additional cable is required to form the hardware synchronization setup.

3.1.2 | Quadrotor frame

The quadrotor frame consists of the mechanical construction, motors, propellers, and the power supply module. The frame is built with carbon fiber tubes and aluminum components, making it light but durable. The whole system is shown in Figure 1. The total weight, including the perception core, is 1.8 kg. Tip-to-tip distance is 33 cm. We use the DJI E310¶ motors and the DJI Takyon Z425-M# ESCs as the lifting system. We use the intelligent battery of DJI Phantom 4 Pro‖ to provide approximately 15 min of flight time. A DJI Lightbridge 2** is used for the implementation of the remote desktop functionality on the iPad.

3.2 | Software architecture

The software architecture of our system is shown in Figure 3. Utilizing ROS as the communication middleware, we distribute computation loads among the two onboard computers under the principle that dense pixelwise operations should be performed on the NVIDIA TX1. The two onboard computers are connected using Ethernet.

3.2.1 | Algorithm pipeline

On the Mini i7 computer, 100 Hz IMU measurements, as well as the 20 Hz synchronized fisheye images, are fused in the visual-inertial state estimator (Section 5) to obtain pose, velocity, and attitude for dense mapping, trajectory planning, and feedback control. The estimator output is at 100 Hz, which is sufficient for feedback control of agile aerial robots. To utilize motion stereo algorithms that are originally designed for pinhole images, we use a region-of-interest (ROI) extraction module (Section 4.2) to crop the fisheye images into two pinhole images to cover the 180-degree horizontal FOV. Pinhole images are then bundled with the estimated pose and sent to the TX1 for depth estimation.

Utilizing the GPUs onboard the NVIDIA TX1, depth images are computed at 10 Hz using dense multiview motion stereo with semiglobal smoothing (Section 6). Depth images are turned into a global map using GPU-accelerated truncated signed distance function (TSDF) fusion, also running at 10 Hz.

The global map is sent back to the Mini i7 computer to assist trajectory planning (Section 7), in which collision-free time parametrized trajectories that guide the robot toward user-specified goals are generated. The trajectory is executed using the attitude and thrust control interface in the DJI A3 flight controller.

Unlike many existing systems that rely on a downward-facing camera for state estimation, our system uses only one camera for all sensing needs. We show through online indoor and outdoor experiments (Section 8) that our minimal sensing setup is sufficient to achieve completely autonomous navigation.

3.2.2 | Remote monitoring and debugging interface

Besides the airborne software, we also utilize the DJI Lightbridge 2 to implement an iOS app for remote monitoring and debugging. The Lightbridge directly streams the HDMI output of the Mini i7 computer to an iPad Pro.†† This enables us to view the desktop in real-time even when the robot is flying. We use the Data Transparent Transmission protocol‡‡ in the Lightbridge to stream virtual keyboard and mouse input from the iPad to the airborne computer. In this way, we are able to directly use the iOS app to run visualization tools such as rviz on the airborne computer. It even enables us to debug and recompile the airborne code while the quadrotor is flying. This setup is independent of any external infrastructure, which makes it very convenient for field tests.

4 | FISHEYE CAMERA

The ultrawide FOV provided by the fisheye camera is the key to safe navigation around the obstacle, but it also comes with a great deal of lens distortion, which must be modeled and corrected. In this section, we discuss how the fisheye is modeled, and the necessary preprocessing steps to enable the use of fisheye images for both visual-inertial state estimation and dense depth estimation.

4.1 | Fisheye camera model

We use the unified sphere model proposed in Ref. 14 and implemented by Ref. 15 to model the fisheye camera. As shown in Figure 4(a), the camera projection model is a function $\pi_c : \mathbb{R}^3 \rightarrow \Omega$, which models the

* https://www.matrix-vision.com/USB2.0-single-board-camera-mvbluefox-mlc.html

† http://www.dji.com/a3

‡ https://www.matrix-vision.com/manuals/mvBlueFOX/HRTC_page_0.html

§ https://developer.dji.com/onboard-sdk/

¶ http://www.dji.com/cn/e310

http://www.dji.com/takyon-z425-m-and-z415-m

‖ http://store.dji.com/product/phantom-4-pro-intelligent-battery-high-capacity

** http://www.dji.com/cn/lightbridge-2

†† http://www.apple.com/ipad-pro/

‡‡ https://developer.dji.com/onboard-sdk/documentation/introduction/data-transparent-transmission.html

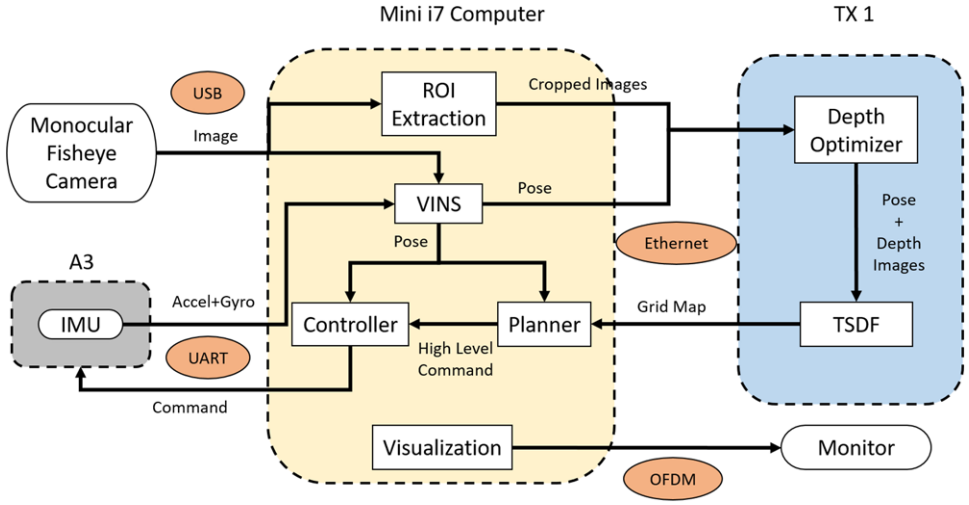
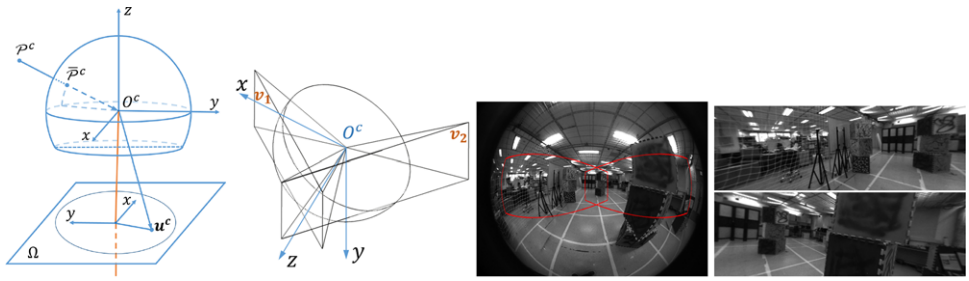


FIGURE 3 System diagram



(a) Fisheye camera model (b) Illustration of ROI (c) ROI on a fisheye image (d) Pinhole images from ROI

FIGURE 4 Illustrations of the camera model (a) and the ROI extraction process. A 3D point $\mathcal{P}^c$ forms a laterally inverted image $\mathbf{u}^c$ by the fisheye lens. Two ROIs that cover the 180-degree horizontal view are set in a fisheye image, as illustrate as v_1 and v_2 in (b), and marked in (c). The two resulting distortion-free pinhole images from the ROI are shown in (d)

relationship between a 3D point and its location on the 2D image plane. The backprojection model $\pi_c^{-1} : \Omega \to \mathbb{R}^3$ is its inverse. In the model, a 3D point $\mathcal{P}^c \in \mathbb{R}^3$ in the camera frame is first projected as a point $\bar{\mathcal{P}}^c$ on the unit sphere, and then transformed onto the 2D image plane as $\mathbf{u}^c$. Lens distortion is handled in the $\pi_c(\cdot)$. This can be mathematically described as

$$\bar{\mathcal{P}}^c = \mathcal{P}^c / ||\mathcal{P}^c||$$
$$\mathbf{u}^c = \pi_c(\bar{\mathcal{P}}^c) \quad (1)$$

The back projection model is $\bar{\mathcal{P}}^c = \pi_c^{-1}(\mathbf{u}^c)$, which maps a point on the 2D image plane back to the unit sphere.

To utilize the full perceptual coverage of the fisheye camera in the monocular visual-inertial state estimator (Section 5), we define the image measurement residual directly on the unit sphere. More details will be given in Section 5.5.3.

4.2 | Region-of-interest (ROI) extraction

Many procedures in multiview dense depth estimation can be highly accelerated by utilizing a 1D scanline search. However, the existence of significant lens distortion makes it hard to directly apply these acceleration techniques on fisheye images. We note that for an aerial robot that mainly undergoes horizontal motion, a large horizontal FOV is the most critical way to ensure safety. To this end, we introduce a region-of-interest (ROI) extraction module to turn a fisheye image into two distortion-free pinhole images that cover a 180-degree horizontal FOV.

We consider a virtual ideal pinhole camera to capture the distortion-free image. The FOV of the virtual camera is manually chosen according to application needs, from which the projection function $\pi_v(\cdot)$ of the virtual camera can be uniquely determined. The orientation of the virtual camera with respect to the fisheye camera frame ($\mathbf{R}_v^c$) is also chosen by the user, as illustrated in Figure 4(b). For a pixel $\mathbf{u}^v$ in the virtual image, we have

$$\mathbf{u}^c = \pi_c(\mathbf{R}_v^c \pi_v^{-1}(\mathbf{u}^v)). \quad (2)$$

Once we obtain the corresponding pixel location in the fisheye image, we use subpixel interpolation to determine the intensity value.

An illustration of the two ROIs that we used in our system is shown in Figure 4(b), with the corresponding fisheye views marked in Figure 4(c). The two distortion-free pinhole images obtained from the ROI are shown in Figure 4(d). The two pinhole images are oriented ± 45 degrees from the physical optical axis. Although in theory we only need the horizontal FOV of each of the pinhole images to be 90 degrees, we set the FOV of both virtual images to be 120 degrees in actual implementation in order to provide redundancy for the center region.

5 | MONOCULAR VISUAL-INERTIAL STATE ESTIMATION

State estimation is the core of autonomous flight. Mapping and path planing require a high accuracy of metric states (position, orientation, and velocity). In this section, we present a monocular tightly coupled sliding window based visual-inertial state estimator, which provides accurate states for the whole system. Since the monocular visual-inertial fusion is a fragile and highly nonlinear procedure, stable feature tracking and good estimator initialization are required. In the following, we will detail the measurement preprocessing, the robust estimator initialization procedure, and the tightly coupled optimization framework. This is a continuation of our previous work.[16,17]

5.1 | Related work

Recently, we have seen a lot of work on visual-inertial estimation with either a monocular camera,[18–20] stereo cameras,[21] or RGB-D cameras.[22] Based on the frameworks, we can divide VINS into two categories: one is a filtering-based algorithm,[18–20,23–26]while the other is a graph optimization or bundle adjustment based method.[16,21,27] The filtering-based algorithm is usually considered to be more efficient in computation. The main drawback is that fixing the linearization points early may lead to sub-optimal results. Via repetitively linearizing past states, the graph optimization-based approaches may achieve better performance, while they require greater computational resources. Mathematically speaking, these methods are the same but are realized in different forms.

For monocular visual-inertial fusion, an accurate initialization of the system is required, but a monocular camera cannot directly provide metric measurements. Our earlier works[16,17,28] proposed an optimization-based linear estimator initialization method. However, it fails in environments where feature depths are distributed throughout a wide range (e.g., outdoor environments) due to the incapability of modeling the sensor noise in the raw projective formulation. Also, our earlier work does not take gyroscope bias into consideration. Another closed-form solution was introduced in Ref. 29. Later, a revision of this closed-form solution was proposed in Ref. 30. This closed-form solution is sensitive to noisy sensor data, since the authors fail to model measurement noise for different inertial integration at different time durations. In Ref. 31, a re-initialization and failure recovery algorithm based on SVO[32] is proposed. In this method, inertial measurements are used first to stabilize the MAV's attitude, then the SVO is launched to stabilize the position. Visual and inertial information are loosely fused by MSF.[26]

One efficient technique dealing with IMU measurement is pre-integration, which avoids repeated integration by a re-parametrization of the relative motion constraints. This algorithm was first proposed in Ref. 33, and it was advanced to consider on-manifold uncertainty propagation in our previous work.[16] Further improvements on incorporating IMU biases and integrating them with a full SLAM framework were proposed in Ref. 34.

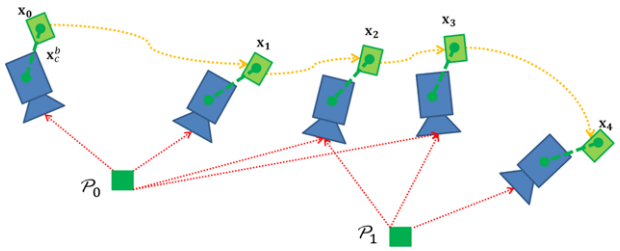


FIGURE 5 An example of the sliding window with five IMU states $\mathbf{x}_k$ and two features $\mathcal{P}_l$. Yellow lines represent the pre-integrated IMU measurements and red lines illustrate visual measurements. Note that there is a constant but unknown camera-IMU extrinsic calibration $\mathbf{x}_c^b$. All aforementioned quantities as well as IMU bias are jointly estimated in our framework

In this paper, to achieve accurate online estimation, we use a tightly coupled optimization-based monocular visual inertial framework with robust initialization. Our VINS is a sparse and feature-based framework, which extracts and tracks features, pre-integrates IMU in the front-end, and jointly minimizes visual and inertial geometry error in the back-end.

5.2 | Sliding window formulation

We take IMU and camera measurements in a fixed time interval for state estimation. This provide good accuracy due to the use of multi-view constraints. It also enjoys constant-time complexity. As shown in Figure 5, several IMU and camera measurements are incorporated in the fixed window. The full state vector in the sliding window is defined as (the transpose is ignored for simplicity of presentation)

$$\begin{aligned}\mathcal{X} &= \left[\mathbf{x}_0, \mathbf{x}_1, \cdots \mathbf{x}_n, \mathbf{x}_c^b, \lambda_0, \lambda_1, \cdots \lambda_m\right]\\ \mathbf{x}_k &= \left[\mathbf{p}_{b_k}^w, \mathbf{v}_{b_k}^w, \mathbf{q}_{b_k}^w, \mathbf{b}_a^b, \mathbf{b}_g^b\right], \quad k \in [0, n]\\ \mathbf{x}_c^b &= \left[\mathbf{p}_c^b, \mathbf{q}_c^b\right],\end{aligned} \tag{3}$$

where $\mathbf{x}_k$ is the kth frame state, which contains position, velocity, and rotation in the world frame and acceleration bias and gyroscope bias in the body frame. Here, we treat the IMU frame as a body frame. The world frame is related to the real world where the gravity is vertical. The world frame will be set after the gravity vector is solved in the initialization procedure. In the whole estimation framework, we use the quaternion to represent the rotation. $\mathbf{x}_c^b$ is the extrinsic parameter, which contains rotation and translation from the camera frame to the IMU frame. n is the number of keyframes in the sliding window, m is the number of features in the sliding window, and λ_l is the inverse depth of the lth feature from its first observation on the unit sphere.

5.3 | Measurement preprocessing

Two kinds of measurements exist in our state estimation framework. One is the images, and the other is the IMU measurements. These two kinds of measurements are preprocessed before incorporated into the estimation. We track and detect features from the images in the preprocessing step. Meanwhile, IMU measurements are pre-

integrated. Note that IMU measurements are not only affected by noise but also bias. So we especially take bias into consideration in IMU pre-integration and the following optimization process.

5.3.1 | Feature processing front-end

For each new image, the existing features are tracked by the KLT sparse optical flow algorithm.[35] Meanwhile, new corner features are detected to maintain a minimum number of features (100–300) in each image. The detector enforces a uniform feature distribution by setting a minimum separation of 20–30 pixels between two neighboring features. Features are projected to a unit sphere using the fisheye camera model before passing into outlier rejection. Outlier rejection is simply performed by the RANSAC step in the fundamental matrix test.

Keyframes are also selected in this step. We have two criteria for the keyframe selection, and one of them is the average parallax. If the average parallax of the tracked features is beyond a certain threshold, we treat this image as a keyframe. Note that not only translation but also rotation can cause parallax; however, features cannot be triangulated in the rotation-only motion. To avoid this situation, we use the IMU propagation result to compensate the rotation when calculating the parallax. Another criterion is the tracking quality. We also treat a new frame as a keyframe if the number of tracked features goes below a certain threshold.

5.3.2 | IMU pre-integration

We denote IMU measurements (angular velocity and acceleration) as $\hat{\boldsymbol{\omega}}_t^b, \hat{\mathbf{a}}_t^b$. These measurements are affected by bias $\mathbf{b}$ and noise $\boldsymbol{\eta}$,

$$\begin{aligned}\hat{\boldsymbol{\omega}}_t^b &= \boldsymbol{\omega}_t^b + \mathbf{b}_g + \boldsymbol{\eta}_g \\ \hat{\mathbf{a}}_t^b &= \mathbf{R}(\mathbf{q}_t^w)^{\mathsf{T}}(\mathbf{a}_t^w + \mathbf{g}^w) + \mathbf{b}_a + \boldsymbol{\eta}_a.\end{aligned} \tag{4}$$

Given two time instants k and $k+1$ that correspond to image frame b_k and b_{k+1}, we can pre-integrate linear acceleration and angular velocity in the local frame b_k:

$$\begin{aligned}\boldsymbol{\alpha}_{b_{k+1}}^{b_k} &= \iint_{t\in[k,k+1]} \boldsymbol{\gamma}_{b_t}^{b_k}\hat{\mathbf{a}}_t dt^2 \\ \boldsymbol{\beta}_{b_{k+1}}^{b_k} &= \int_{t\in[k,k+1]} \boldsymbol{\gamma}_{b_t}^{b_k}\hat{\mathbf{a}}_t dt \\ \boldsymbol{\gamma}_{b_{k+1}}^{b_k} &= \int_{t\in[k,k+1]} \boldsymbol{\gamma}_{b_t}^{b_k} \otimes \begin{bmatrix} 0 \\ \frac{1}{2}\hat{\boldsymbol{\omega}}_t \end{bmatrix} dt.\end{aligned} \tag{5}$$

In the above formula, $\otimes$ denotes the quaternion multiplication operation. It can be seen that the pre-integration part can be obtained solely with IMU measurements within $[k, k+1]$ since it is independent of states (pose and velocity).

5.4 | Initialization

Since tightly coupled visual-inertial optimization is nonlinear, we need a good initial guess to bootstrap the whole system. We cannot directly use the IMU pre-integration result as the initial guess since we do not know the initial attitude and velocity. Also, the IMU measurement is affected by unknown biases, especially gyroscope bias, which will dramatically affect the estimation.

We adopt a loosely coupled sensor fusion method to get the initial values. We find that vision-only SLAM, or Structure from Motion (SfM), has a good property of initialization. In most cases, a visual-only system can bootstrap itself by a derived initial guess from the relative motion method, such as the Eight-point,[36] Five-point,[37] homogeneous, and fundamental matrix. Through loosely coupled alignment between the visual-only SfM and IMU metric pre-integration information, we can roughly recover the scale, gravity, velocity, and gyroscope bias. This is sufficient for bootstrapping a nonlinear system.

First, we construct the visual structure of our sliding window, which contains up-to-scale pose and feature position. Secondly, we align visual and IMU information to calibrate the IMU gyroscope bias, scale, velocity, and gravity vector. Though the first step is nonlinear, it can bootstrap itself and is quite stable. The second step is in linear form.

The extrinsic parameter $\mathbf{x}_c^b$ between camera and IMU is needed in the initial step. Note that we only need an initial guess about the extrinsic parameter here, and accurate estimation will be carried out in nonlinear optimization.

5.4.1 | Up-to-scale visual SfM in sliding window

In this step, we try to construct the vision-only structure (up-to-scale camera pose and feature position) within the window.

We first choose two keyframes that contain sufficient feature parallax in the sliding window. Next, we use the five-point method[37] to recover the relative rotation and up-to-scale translation between these two frames, then we fix the scale of translation and triangulate the features observed in these two frames. Based on these triangulated features, the Perspective-n-Point (PnP) method* is performed to estimate all frame poses in the sliding window.

A global full Bundle Adjustment[38] is then applied to minimize the total re-projection error of all feature observations in all frames. After that, we get all keyframe poses $(\bar{\mathbf{p}}_{c_i}^{c_0}, \mathbf{q}_{c_i}^{c_0})$ and feature positions. Here, $\bar{(\cdot)}$ denotes up-to-scale variables. We write all poses in the local frame $(\cdot)^{c_0}$, which is the first camera frame in the window. Since the extrinsic parameter $(\mathbf{p}_c^b, \mathbf{q}_c^b)$ between camera frame and IMU (body) frame is known, all variables are transformed into the IMU frame,

$$\begin{aligned}\mathbf{q}_{b_i}^{b_0} &= \mathbf{q}_c^b \otimes \mathbf{q}_{c_i}^{c_0} \\ s\bar{\mathbf{p}}_{b_i}^{b_0} &= s\mathbf{q}_c^b\bar{\mathbf{p}}_{c_i}^{c_0} + \mathbf{p}_c^b\end{aligned} \tag{6}$$

s is the unknown scale, which will be solved in the next section.

5.4.2 | Visual-inertial alignment

IMU Bias Initialization

Assume the IMU gyroscope bias $\mathbf{b}_g$ is constant in the current window. Considering two consecutive frames b_k and b_{k+1} in the window, we have the relative rotation $\mathbf{q}_{b_k}^{b_0}$ and $\mathbf{q}_{b_{k+1}}^{b_0}$ from the visual structure,

* http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#solvepnp

as well as the rotation propagation result $\hat{\gamma}^{b_k}_{b_{k+1}}$ from the IMU pre-integration. We estimate the gyroscope bias by minimizing the error between these two terms:

$$\min_{\mathbf{b}_g} \sum_{k\in\mathcal{B}} \left\| \mathbf{q}^{b_0}_{b_{k+1}}{}^{-1} \otimes \mathbf{q}^{b_0}_{b_k} \otimes \gamma^{b_k}_{b_{k+1}} \right\|^2$$

$$\gamma^{b_k}_{b_{k+1}} \approx \hat{\gamma}^{b_k}_{b_{k+1}} \otimes \begin{bmatrix} 1 \\ \frac{1}{2} \frac{\partial \gamma^{b_k}_{b_{k+1}}}{\partial \mathbf{b}_g} \mathbf{b}_g \end{bmatrix}, \tag{7}$$

where $\mathcal{B}$ indexes the IMU measurement in the window. By solving this least-squares problem, we can get the estimation of $\mathbf{b}_g$. Then we update $\hat{\boldsymbol{\alpha}}^{b_k}_{b_{k+1}}, \hat{\boldsymbol{\beta}}^{b_k}_{b_{k+1}}$ with respect to $\mathbf{b}_g$.

As for the accelerometer bias, it is hard to solve it at the initialization procedure, since sufficient rotation movements are needed to distinguish the accelerometer bias and the gravity, when we are solving $\mathbf{g}^{b_0}$ at the same time. However, giving a coarse initial guess is adequate because we will continuously refine the bias after the initialization. Hence, we treat accelerometer bias $\mathbf{b}_a$ as zero in the initialization step.

Velocity, Gravity Vector, and Metric Scale Initialization

We define the variables that we would like to estimate as

$$\mathcal{X}_I = \left[\mathbf{v}^{b_0}_{b_0}, \mathbf{v}^{b_0}_{b_1}, \cdots \mathbf{v}^{b_0}_{b_n}, \mathbf{g}^{b_0}, s \right], \tag{8}$$

where s is the scale parameter that aligns the visual structure to the actual metric scale implicitly provided by IMU measurements. Based on the Newton kinematic, we can get the equation in the following form:

$$\hat{\mathbf{z}}^{b_k}_{b_{k+1}} = \begin{bmatrix} \hat{\boldsymbol{\alpha}}^{b_k}_{b_{k+1}} \\ \hat{\boldsymbol{\beta}}^{b_k}_{b_{k+1}} \end{bmatrix} = \mathbf{H}^{b_k}_{b_{k+1}} \mathcal{X}_I + \mathbf{n}^{b_k}_{b_{k+1}}$$

$$\approx \begin{bmatrix} -\mathbf{q}^{b_k}_{b_0} \Delta t_k & \mathbf{0} & \frac{1}{2}\mathbf{q}^{b_k}_{b_0} \Delta t_k^2 & \mathbf{q}^{b_k}_{b_0} (\bar{\mathbf{p}}^{b_0}_{b_{k+1}} - \bar{\mathbf{p}}^{b_0}_{b_k}) \\ -\mathbf{q}^{b_k}_{b_0} & \mathbf{q}^{b_k}_{b_0} & \mathbf{q}^{b_k}_{b_0} \Delta t_k & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{v}^{b_0}_{b_k} \\ \mathbf{v}^{b_0}_{b_{k+1}} \\ \mathbf{g}^{b_0} \\ s \end{bmatrix}. \tag{9}$$

In the above formula, $\mathbf{q}^{b_0}_{b_k}, \bar{\mathbf{p}}^{b_0}_{b_k}, \bar{\mathbf{p}}^{b_0}_{b_{k+1}}$ are obtained from the visual structure with respect to body frame $(\cdot)^{b_0}$. Δt_k is the time interval between two consecutive keyframes. By solving the following least-squares problem:

$$\min_{\mathcal{X}_I} \sum_{k\in\mathcal{B}} \left\| \hat{\mathbf{z}}^{b_k}_{b_{k+1}} - \mathbf{H}^{b_k}_{b_{k+1}} \mathcal{X}_I \right\|^2, \tag{10}$$

we can get the velocities in every local frame, and the gravity vector in the visual base frame $(\cdot)^{b_0}$, as well as the scale parameter. The translational components $\bar{\mathbf{p}}^{b_i}$ from the visual structure will be scaled to the metric units. We rotate all variables from frame $(\cdot)^{b_0}$ to the world frame $(\cdot)^w$ where the gravity vector is vertical. At this point, the initialization procedure is completed and these metric values will be fed for a tightly coupled nonlinear visual-inertial estimator.

5.5 | Tightly coupled visual-inertial localization

After state initialization, we proceed with a sliding window nonlinear estimator for high-accuracy state estimation. This is an extension of our earlier work[16,17] by including IMU bias calibration in the nonlinear optimization framework.

5.5.1 | Formulation

We minimize the sum of the Mahalanobis norm of all measurement residuals to obtain a maximum of *a posteriori* estimation:

$$\min_{\mathcal{X}} \left\{ \left\| \mathbf{r}_p - \mathbf{H}_p \mathcal{X} \right\|^2 + \sum_{k\in\mathcal{B}} \left\| r_{\mathcal{B}}(\hat{\mathbf{z}}^{b_k}_{b_{k+1}}, \mathcal{X}) \right\|^2_{\mathbf{P}^{b_k}_{b_{k+1}}} + \sum_{(l,j)\in\mathcal{C}} \left\| r_{\mathcal{C}}(\hat{\mathbf{z}}^{c_j}_{l}, \mathcal{X}) \right\|^2_{\mathbf{P}^{c_j}_{l}} \right\}, \tag{11}$$

where $r_{\mathcal{B}}(\hat{\mathbf{z}}^{b_k}_{bk+1}, \mathcal{X})$ and $r_{\mathcal{C}}(\hat{\mathbf{z}}^{c_j}_{l}, \mathcal{X})$ are measurement residuals for the IMU and camera, respectively. $\mathcal{B}$ is the set of all IMU measurements, and $\mathcal{C}$ is the set of feature observations in the corresponding frame. Corresponding measurement models are defined in Sections 5.5.2 and 5.5.3. $\{\mathbf{r}_p, \mathbf{H}_p\}$ is the prior information, which will be discussed in Section 5.5.4. We use an error-state representation to linearize the nonlinear system (11) and solve it via the Gauss-Newton algorithm.

5.5.2 | IMU measurement model

Following the kinematics theory, the residual of a pre-integrated IMU measurement can be defined as

$$r_{\mathcal{B}}(\hat{\mathbf{z}}^{b_k}_{b_{k+1}}, \mathcal{X}) = \begin{bmatrix} \delta\boldsymbol{\alpha}^{b_k}_{b_{k+1}} \\ \delta\boldsymbol{\beta}^{b_k}_{b_{k+1}} \\ \delta\boldsymbol{\gamma}^{b_k}_{b_{k+1}} \\ \delta\mathbf{b}_a{}^{b_k}_{b_{k+1}} \\ \delta\mathbf{b}_g{}^{b_k}_{b_{k+1}} \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{q}^{b_k}_w (\mathbf{p}^w_{b_{k+1}} - \mathbf{p}^w_{b_k} + \mathbf{g}^w \Delta t^2/2) - \mathbf{v}^{b_k}_{b_k} \Delta t - \hat{\boldsymbol{\alpha}}^{b_k}_{b_{k+1}} \\ \mathbf{q}^{b_k}_w (\mathbf{q}^w_{b_{k+1}} \mathbf{v}^{b_{k+1}}_{b_{k+1}} + \mathbf{g}^w \Delta t) - \mathbf{v}^{b_k}_{b_k} - \hat{\boldsymbol{\beta}}^{b_k}_{b_{k+1}} \\ 2\left[\hat{\gamma}^{b_k^{-1}}_{b_{k+1}} \otimes \mathbf{q}^{b_w^{-1}}_{b_k} \otimes \mathbf{q}^{b_w}_{b_{k+1}} \right]_{xyz} \\ \mathbf{b}_a{}^{b_{k+1}}_{b_{k+1}} - \mathbf{b}_a{}^{b_k}_{b_k} \\ \mathbf{b}_g{}^{b_{k+1}}_{b_{k+1}} - \mathbf{b}_g{}^{b_k}_{b_k} \end{bmatrix}, \tag{12}$$

where $[\cdot]_{xyz}$ extracts the vector part of the quaternion $\mathbf{q}$, which is the approximation of the error state representation. $\left[\hat{\boldsymbol{\alpha}}^{b_k}_{b_{k+1}}, \hat{\boldsymbol{\beta}}^{b_k}_{b_{k+1}}, \hat{\gamma}^{b_k}_{b_{k+1}} \right]^{\mathsf{T}}$ is the pre-integrated IMU measurement using only noisy accelerometer and gyroscope measurements, which is related to accelerometer and gyroscope bias and independent of initial velocity and attitude.

The covariance matrix $\mathbf{P}^{b_k}_{b_{k+1}}$ can then be calculated by first-order discrete-time propagation within the time interval $[k, k+1]$.

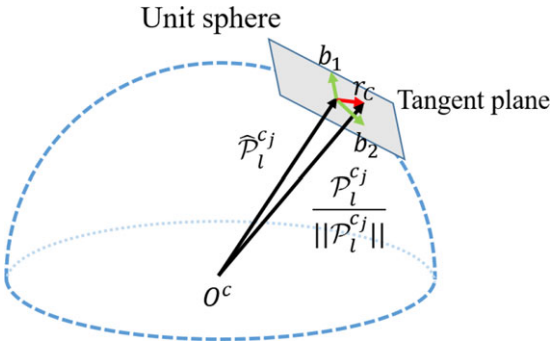


FIGURE 6 An illustration of the camera measurement residual on the unit sphere. $\hat{\mathcal{P}}_l^{c_j}$ is the ray vector of *l*th feature observed in the *j*th frame , $\mathcal{P}_l^{c_j}$ is the transformed ray vector from the *i*th frame. The residual is defined on the tangent plane of $\hat{\mathcal{P}}_l^{c_j}$

5.5.3 | Camera measurement model

To utilize the benefit of a large FOV camera, we define the camera measurement residual on a unit sphere as proposed in Ref. 39, which matches the fisheye camera model. As shown in Figure 6, the camera residual for the observation of the *l*th feature in the *j*th image is defined as

$$
\begin{aligned}
r_C(\hat{\mathbf{z}}_l^{c_j}, \mathcal{X}) &= \begin{bmatrix}\mathbf{b}_1 & \mathbf{b}_2\end{bmatrix}^T \cdot \left(\hat{\mathcal{P}}_l^{c_j} - \frac{\mathcal{P}_l^{c_j}}{\|\mathcal{P}_l^{c_j}\|}\right) \\
\hat{\mathcal{P}}_l^{c_j} &= \pi_c^{-1}\left(\begin{bmatrix}\hat{u}_l^{c_j} \\ \hat{v}_l^{c_j}\end{bmatrix}\right) \\
\mathcal{P}_l^{c_j} &= \begin{bmatrix}x_l^{c_j} \\ y_l^{c_j} \\ z_l^{c_j}\end{bmatrix} = \mathbf{T}_c^{b^{-1}} \cdot \mathbf{T}_{b_j}^{w^{-1}} \cdot \mathbf{T}_{b_i}^{w} \cdot \mathbf{T}_c^b \cdot \frac{1}{\lambda_l} \cdot \pi_c^{-1}\left(\begin{bmatrix}u_l^{c_i} \\ v_l^{c_i}\end{bmatrix}\right),
\end{aligned}
\tag{13}
$$

where $[u_l^{c_i}\ v_l^{c_i}]^T$ is the noiseless first observation of the *l*th feature that happens in the *i*th image. $[\hat{u}_l^{c_j}\ \hat{v}_l^{c_j}]^T$ is the observation of the same feature in the *j*th image. $\mathbf{T} = \begin{pmatrix}\mathbf{R} & \mathbf{p} \\ \mathbf{0} & 1\end{pmatrix}$, which is the homogeneous matrix of transformation. To simply the representation, we omit the homogeneous term in the above equations. $\mathbf{T}_c^b$ is the transformation from the camera frame to the IMU (body) frame, and its inverse transforms from the IMU frame to the camera frame. $\mathbf{T}_{b_i}^w$ transforms from the *i*th IMU frame to the world frame. $\pi_c^{-1}(\cdot)$ is the backprojection function, which outputs the unit vector in 3D space. Since the degree of freedom of the vision residual is two, we project the unit vector residual into the tangent plane, which guarantees the right degree of freedom. $\mathbf{b}_1, \mathbf{b}_2$ are two arbitrarily selected orthogonal bases that span the tangent plane of $\hat{\mathcal{P}}_l^{c_j}$.

5.5.4 | Marginalization

To bound the computational complexity of graph optimization-based methods, marginalization is incorporated. We selectively marginalize out IMU states $\mathbf{x}_k$ and features λ_l from the sliding window while converting measurements corresponding to the marginalized states into a prior.

In our previous work,[16,17] we use the two-way-marginalization scheme to selectively remove recent or old states based on the scene parallax test. A new frame is added to the sliding window if it is a keyframe, and the oldest frame states are marginalized out with its corresponding measurement. Otherwise, if a non-keyframe comes, we marginalize out the second newest frame. This marginalization scheme can keep spatial keyframes in the window while bounding the uncertainty for pre-integrated IMU measurements.

However, one drawback of our previous strategy is that marginalizing out the second newest frame generates a dense prior, which will destroy the sparsity of the system, increasing the computational complexity. To avoid this, we slightly modify the strategy on marginalizing the second newest frame. When a non-keyframe comes, instead of marginalizing out all measurement, we discard the visual measurements and keep the IMU measurements corresponding to the second newest frame in the window. Although we drop some visual measurements, these measurements are not significant since they correspond to small motion, which does not affect the visual structure. One potential drawback is that we extend the time interval of preintegration, which will induce large covariance into the sliding window. However, this strategy maintains the sparsity of the system, making it efficient and work well on a computation-limited platform.

We construct a new prior based on marginalized measurements related to the removed state. The marginalization is carried out using the Schur complement. Intuitively, by marginalization, important information of the removed states is kept and computation complexity is bounded.

5.6 | IMU propagation for feedback control

Note that the IMU measurements come at a much higher rate than visual measurements. The frequency of our nonlinear optimization estimator is limited by visual measurements. To benefit the performance for real-time control, the outputs of the estimator are directly propagated with the newest IMU measurements, which serves as the high-frequency feedback in the control loop.

6 | MONOCULAR DENSE MAPPING

As the core of the autonomous quadrotor navigation, the mapping module provides a perception ability to reconstruct the surrounding environment. This reconstruction is the backbone for safe and efficient navigation around obstacles. The method used in this work in the continuation of our preliminary results.[40] We use temporal cost aggregation from multiple images from the monocular to eliminate baseline limitation. This is followed by semiglobal smoothing for outlier removal and depth propagation in textureless environments. We utilize the GPUs onboard the NVIDIA TX1 and implement our algorithm with CUDA.* Local depth images are fused using the truncated signed distance field (TSDF) to provide a global map that is directly used for trajectory planning.

* https://developer.nvidia.com/cuda-toolkit

6.1 | Related work

Extensive scholarly work exists in the field of dense mapping, and a large variety of sensors, such as RGB-D cameras, stereo cameras, and monocular cameras, are used. In a RGB-D camera setting, KinectFusion[41] simultaneously solves the localization and mapping problem. It uses the iterative closest point (ICP) to solve the camera pose, then reconstructs a global map by TSDF fusion. Autonomous flight using a RGB-D sensors is proposed in Refs. 42 and [43]. With a known mesh map constructed by a Kinect, the authors use direct semidense tracking for pose estimation. However, the RGB-D sensors have their own limitation, which severely limits their usage in outdoor environments.

A spatial stereo camera is another popular sensing configuration. A semiglobal matching stereo method has been presented in Ref. 44. By combining pixelwise cost and smoothness constraints, the author proposes a global energy function to solve the disparity map and optimizes its approximated version by dynamic programming. The stereo matching cost can be further improved using machine learning-based approaches, as proposed in Ref. 45. A spatial stereo can easily suffer from calibration issues as it depends on known camera extrinsic parameters. Moreover, the baseline requirement of stereo-based methods fundamentally limits its capability for small-scale platforms.

For the monocular setting, Ref. 46 presents a self-calibrating method that jointly solves the camera's intrinsic, 3D landmark points together with camera poses in a bundle adjustment framework. The authors generate a fine depth image under small motion conditions by a novel matching cost definition that takes the intensity and gradient variances among candidate depths through a set of images. This algorithm shows impressive depth images, but its time consumption (10 min) for each depth image estimation is unacceptable for real-time quadrotor applications. DTAM[47] combines a data term and a regularized term to form a global energy function. By using a primal-dual approach, DTAM efficiently optimizes the problem through total variation. A similar work, REMODE,[5] also uses a primal-dual approach and optimizes the problem through total variation but combining Bayesian estimation. However, both of them requires desktop-grade GPUs for real-time usage. Semidense mapping[9] is another line of research. Between feature-based sparse mapping and pixel-based dense mapping, the semidense method chooses the pixels with high gradients, which makes the whole framework run real-time on CPU. Nevertheless, such pixel density is not enough for safe navigation.

6.2 | Depth estimation

As discussed in Section 4.2 and illustrated in Figure 4(b), two distortion-free virtual pinhole images are extracted from a fisheye image in order to provide a 180 degrees horizontal FOV. These two images are processed in exactly the same algorithm pipeline for dense mapping. For brevity, we only present the algorithm flow for one of them.

Our depth estimation is done with respect to keyframes. For each keyframe I_r, multiple measurement frames I_m are used for depth update. Only one keyframe exists at any time, and keyframes are switched using a distance metric. A frame is considered as a new keyframe if the pose translation between this frame and the current keyframe is larger than a distance threshold ζ (Figure 7). Inspired by the plane sweeping algorithm,[48,49] we sample a number of candidate virtual planes at different depths for each pixel in the keyframe. These planes are perpendicular to the z-axis of the canonical viewpoint. Each pixel in the keyframe is first projected to a virtual plane, then backprojected to a measurement frame. By measuring the intensity difference, we obtain a similarity cost for each pixel in each depth. We collect all costs and organize them in a 3D cost volume.

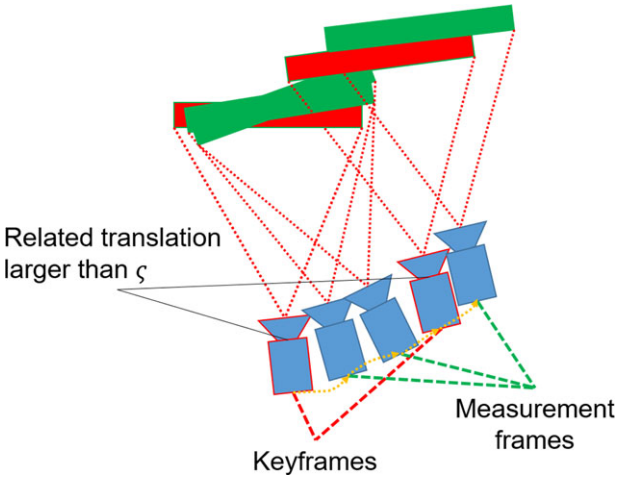


FIGURE 7 A frame with large enough pose translation would be considered as a new keyframe, otherwise it will act as a measurement frame for cost aggregation

When multiple measurement images exist, we may obtain a similarity cost volume for each of them. Since all cost volumes are conditioned on the reference image I_r, we may aggregate all cost volumes into a single one to reduces its sensitivity to noise and outliers. This process is accelerated by parallel implementation using GPU. Intuitively, for each pixel, its actual depth is the one that corresponds to the smallest similarity cost in the aggregated cost volume.

A semiglobal smoothing[44] is then applied for outlier removal and depth interpolation of textureless regions. Parabola fitting is used as the last step for depth refinement.

6.2.1 | Keyframe selection

The motivation for setting distance thresholds is to ensure each keyframe we selected has sufficient information to provide a reliable depth image.

In our previous work,[40] the keyframe was selected by the VINS framework automatically. However, there exist two limitations:

1. As we mentioned in Section 5.3.1, the parallax check in VINS is to select a keyframe with enough parallax. However, this keyframe criterion cannot be configured individually to satisfy the requirement of dense mapping.
2. The sweeping planes are pre-parametrized for recognizing the nearby depth, which means the range of depth awareness is fixed. But we check the average parallax in the VINS where the features far away from the camera are calculated, and we reduced the

average parallax. Considering the situation of large-scale environments, the metric distance of keyframes will become extremely large so that the nearby obstacles will not be observed because of the overflow disparity for the pixel in the near depth.

As an improvement, in this work we use two thresholds to control the keyframe selection. The first threshold ι is used to counter hovering, when we have not enough information for depth awareness. If the translation distance is less than the first threshold, we will only do the aggregation step. The second threshold ζ is parametrized for the keyframe selection. As we mentioned before, this keyframe threshold is configured to adapt the nearest sampling plane setting. Empirically, we treat this threshold large enough to produce keyframes with sufficient information for depth extraction, yet small enough to avoid occlusion and the depth unobservability of near planes.

6.2.2 | GPU-accelerated temporal cost aggregation

We sample L planes with inverse depth λ_k, where k is an integer in the range of $[0, L-1]$. The depth of the kth enumerated plane is $d_k = \frac{1}{k \cdot \lambda_{min}}$, where $d_{min} = \frac{1}{(l-1)\cdot\lambda_{min}}$ is the nearest plane and $d_{max} = \frac{1}{0\cdot\lambda_{min}} = \infty$ is the plane at infinity. Given depth, the backprojection procedure of a 2D pixel $\mathbf{u}$ to a 3D point $\mathcal{P}^w$ in a world frame can be expressed as

$$\mathcal{P}^w = \mathbf{R}_r^w \mathbf{R}_v^c (d \cdot K^{-1} \cdot \begin{bmatrix} \mathbf{u} \\ 1 \end{bmatrix}) + \mathbf{p}_r^w \tag{14}$$

where K is the intrinsic of the virtual pinhole ROI image, which can be uniquely determined by its FOV. The corresponding 2D pixel $\mathbf{u}'$ in measurement frame I_m is computed by perspective projection:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = K(\mathbf{R}_m^w \mathbf{R}_v^c)^T (\mathcal{P}^w - \mathbf{p}_m^w) \tag{15}$$

$$\mathbf{u}' = \begin{bmatrix} \frac{x'}{z'} \\ \frac{y'}{z'} \end{bmatrix}$$

The sum of absolute difference (SAD) is used to calculate the similarity cost. A fixed size of a 3×3 patch is applied. Hence the cost can be expressed as

$$E_{SAD}(\mathbf{u}, k) = \sum_{m \in \mathcal{M}} \sum_{\mathbf{u}_r \in \mathcal{N}(\mathbf{u})} \left| I_r(\mathbf{u}_r) - I_m(\mathbf{u}') \right|, \tag{16}$$

where $\mathcal{M}$ is the set of measurement frames. $\mathcal{N}(\mathbf{u})$ are the neighbor pixels around $\mathbf{u}$.

To parallelize arithmetic operations on GPU, Eqs. (14) and (15) can be combined as

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = K(\mathbf{R}_m^w \mathbf{R}_v^c)^T (\mathbf{R}_r^w \mathbf{R}_v^c (d \cdot K^{-1} \cdot \begin{bmatrix} \mathbf{u} \\ 1 \end{bmatrix}) + \mathbf{p}_r^w - \mathbf{p}_m^w) \tag{17}$$

$$= \frac{1}{k\lambda_{min}} K(\mathbf{R}_m^w \mathbf{R}_v^c)^T \mathbf{R}_r^w \mathbf{R}_v^c K^{-1} \begin{bmatrix} \mathbf{u} \\ 1 \end{bmatrix} + K(\mathbf{R}_m^w \mathbf{R}_v^c)^T (\mathbf{p}_r^w - \mathbf{p}_m^w) \tag{18}$$

$$= \frac{1}{k\lambda_{min}} \mathbf{H} \begin{bmatrix} \mathbf{u} \\ 1 \end{bmatrix} + \mathbf{J} \tag{19}$$

$$= \frac{1}{k\lambda_{min}} \begin{bmatrix} -\mathbf{h}_1- \\ -\mathbf{h}_2- \\ -\mathbf{h}_3- \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ 1 \end{bmatrix} + \begin{bmatrix} J_1 \\ J_2 \\ J_3 \end{bmatrix} \tag{20}$$

$$= \begin{bmatrix} \frac{\mathbf{h}_1 \begin{bmatrix} \mathbf{u}^T & 1 \end{bmatrix}^T}{k\lambda_{min}} + J_1 \\ \frac{\mathbf{h}_2 \begin{bmatrix} \mathbf{u}^T & 1 \end{bmatrix}^T}{k\lambda_{min}} + J_2 \\ \frac{\mathbf{h}_3 \begin{bmatrix} \mathbf{u}^T & 1 \end{bmatrix}^T}{k\lambda_{min}} + J_3 \end{bmatrix} \tag{21}$$

$$\mathbf{u}' = \begin{bmatrix} \left(\frac{\mathbf{h}_1 \begin{bmatrix} \mathbf{u}^T & 1 \end{bmatrix}^T}{k\lambda_{min}} + J_1 \right) / \left(\frac{\mathbf{h}_3 \begin{bmatrix} \mathbf{u}^T & 1 \end{bmatrix}^T}{k\lambda_{min}} + J_3 \right) \\ \left(\frac{\mathbf{h}_2 \begin{bmatrix} \mathbf{u}^T & 1 \end{bmatrix}^T}{k\lambda_{min}} + J_2 \right) / \left(\frac{\mathbf{h}_3 \begin{bmatrix} \mathbf{u}^T & 1 \end{bmatrix}^T}{k\lambda_{min}} + J_3 \right) \end{bmatrix}. \tag{22}$$

A pixel location in the measurement frame $\mathbf{u}'$ is found by first backprojecting a pixel in the keyframe $\mathbf{u}$ to a 3D point that is defined by the candidate plane, then reprojecting this 3D point onto the measurement frame. For every pixel $\mathbf{u}$ in the keyframe, we adopt this procedure for its neighbor pixel repeatedly to calculate the similarity cost as shown in Eq. (16). The cost volume E_{SAD} is computed by aggregating the similarity cost from multiple measurement images captured at different time instances. To increase the intensity accuracy of the floating point pixel location $\mathbf{u}_m$, bilinear interpolation is used. This will happen automatically during GPU calculation using texture memory, which does not consume any extra computational resources.

6.2.3 | Semiglobal optimization

Temporal cost aggregation aims to decrease the sensitivity of image noise by using multiple measurements. But still, the simple winner-takes-all strategy will not get reliable results since mismatch can easily occur due to repetitive patterns. In addition, in textureless regions, the cost volume will contain a similar cost among different depths, which leads to ambiguity in depth estimation. To this end, a smoothing-based optimization method is adopted in our depth estimator.

Since we use enumerated planes to represent the depth image, the depth optimization problem now becomes a plane index optimization problem. Denote $\boldsymbol{\mathcal{K}}$ as the plane index map that we try to optimize. The global energy function $E(\boldsymbol{\mathcal{K}})$ which combines the pixelwise cost and smoothness constraints is expressed as

$$E(\boldsymbol{\mathcal{K}}) = \sum_{\mathbf{u} \in \Omega} \Bigg(E_{SAD}(\mathbf{u}, \boldsymbol{\mathcal{K}}_\mathbf{u}) + P_1 \cdot \sum_{\mathbf{u}_q \in \mathcal{N}(\mathbf{u})} T[|\boldsymbol{\mathcal{K}}_\mathbf{u} - \boldsymbol{\mathcal{K}}_{\mathbf{u}_q}| = 1] + P_2 \cdot \sum_{\mathbf{u}_q \in \mathcal{N}(\mathbf{u})} T[|\boldsymbol{\mathcal{K}}_\mathbf{u} - \boldsymbol{\mathcal{K}}_{\mathbf{u}_q}| > 1] \Bigg), \tag{23}$$

where Ω is the keyframe image domain and $\boldsymbol{\mathcal{K}}_\mathbf{u}$ is the enumerated plane index at pixel $\mathbf{u}$. $T[\cdot]$ is the indicator function, which is 1 if the expression inside is true, and 0 otherwise. The energy function for each pixel consists of three kinds of terms: one data term directly extracted from the

cost volume and two regularization terms. The first regularization term P_1 penalizes the energy by neighbor pixels with which the enumerated depth difference is only 1. The second regularization term gives a higher penalty P_2 to larger depth differences.

Since the global minimization is a NP-complete problem that cannot be solved in polynomial time, a semiglobal matching (SGM) is proposed in Ref. 44. In this work, we use four-path SGM as it gives a good tradeoff between speed and accuracy.

$$S(\mathcal{K}) = \sum_{\mathbf{u}\in\Omega}\sum_{\mathbf{r}} L_{\mathbf{r}}(\mathbf{u}, \mathcal{K}_{\mathbf{u}}), \mathbf{r} \in \left\{ [0\ 1]^T, [1\ 0]^T, [0\ -1]^T, [-1\ 0]^T \right\} \quad (24)$$

$$L_{\mathbf{r}}(\mathbf{u}, \mathcal{K}_{\mathbf{u}}) = E_{SAD}(\mathbf{u}, \mathcal{K}_{\mathbf{u}}) + \min \begin{Bmatrix} L_{\mathbf{r}}(\mathbf{u}-\mathbf{r}, \mathcal{K}_{\mathbf{u}}) \\ L_{\mathbf{r}}(\mathbf{u}-\mathbf{r}, \mathcal{K}_{\mathbf{u}}-1) + P_1 \\ L_{\mathbf{r}}(\mathbf{u}-\mathbf{r}, \mathcal{K}_{\mathbf{u}}+1) + P_1 \\ \min_i \left\{ L_{\mathbf{r}}(\mathbf{u}-\mathbf{r}, i) + P_2 \right\} \end{Bmatrix} - \min_j L_{\mathbf{r}}(\mathbf{u}-\mathbf{r}, j), \quad (25)$$

where the last term $\min_j L_{\mathbf{r}}(\mathbf{u}-\mathbf{r}, j)$ is used to reduce the numerical value since it is smaller than the sum of the first two terms, and it is the same if $\mathbf{u}$ and $\mathbf{r}$ do not change, which guarantees that $L_{\mathbf{r}}(\mathbf{u}, \mathcal{K}_{\mathbf{u}})$ would not be affected. Hence, the problem is simplified as a 1D problem along the direction $\mathbf{r}$. The approximated cost function can be solved in parallel by using GPU. The pseudocode is shown in Algorithm 1 and an intuitive illustration of SGM computing is presented in Figure 8.

ALGORITHM 1 Semiglobal matching on GPU per block

procedure SGM(E_{SAD}, S, k)
  $S(k) \leftarrow E_{SAD}(k)$
  $Lri_{min} \leftarrow \min_i E_{SAD}(i)$
  synchronize_threads
  if $Lri_{min} =$ undefined **then**
    $E_{SAD}(k) \leftarrow 0$
    $S(k) \leftarrow 0$
    $Lrk_{prev}(k) \leftarrow 0$
    $Lri_{min_{prev}}(k) \leftarrow 0$
  else
    $S(k) \leftarrow S(k) + E_{SAD}(k)$
    $Lrk_{prev}(k) \leftarrow E_{SAD}(k)$
    $Lri_{min_{prev}}(k) \leftarrow E_{SAD}(k)$
  end if
  for $\overline{p} \in$ Scanline direction **do**
    $S(k) \leftarrow E_{SAD}(k)$
    $Lri_{min} \leftarrow \min_i E_{SAD}(i)$
    if $Lri_{min} =$ undefined **then**
      $E_{SAD}(k) \leftarrow 0$
    end if
    $G \leftarrow |I(\overline{p}) - I(\overline{p} - r)|$
    if $G \leq G_\tau$ **then**
      $P_1 \leftarrow P_1' \cdot Q_1$
      $P_2 \leftarrow P_2' \cdot Q_2$
    else
      $P_1 \leftarrow P_1'$
      $P_2 \leftarrow P_2'$
    end if
    $val \leftarrow \min(Lrk_{prev}(k), Lri_{min_{prev}}(k) + P_2, Lrk_{prev}(k-1) + P_1, Lrk_{prev}(k+1) + P_1)$
    $val \leftarrow E_{SAD}(k) + val - Lri_{min_{prev}}(k)$
    synchronize_threads
    if $Lri_{min} =$ undefined **then**
      $S(k) \leftarrow 0$
    else
      $S(k) \leftarrow S(k) + val$
    end if
  end for
end procedure

Usually, a pixel with a higher gradient gains higher probability of the depth discontinuity. To support this edge-aware smoothing, we adjust the penalties P_1 and P_2 by means of the pixel gradient G along the SGM scanning direction:

$$P_1 = \begin{cases} P'_1 \cdot Q_1 & \text{if } G <= G_\tau \\ P'_1 & \text{otherwise} \end{cases} \quad P_2 = \begin{cases} P'_2 \cdot Q_2 & \text{if } G <= G_\tau \\ P'_2 & \text{otherwise,} \end{cases} \tag{26}$$

where G_τ is a certain gradient threshold, and P'_1, P'_2, Q_1, Q_2 are constant parameters. Note that Q_1 and Q_2 should be set greater than 1.

6.2.4 | Postprocessing of depth images

After semiglobal optimization, a simple winner-takes-all strategy can be used to obtain the depth by choosing the lowest cost $S(\boldsymbol{\mathcal{K}})$ at each pixel. However, since depth is sampled at a discrete setting, we use a parabola interpolation to obtain a better depth, which is the extrema point between two depth samples,

$$f(\boldsymbol{\mathcal{K}}_\mathbf{u}) = S(\mathbf{u}, \boldsymbol{\mathcal{K}}_\mathbf{u}) = a\boldsymbol{\mathcal{K}}_\mathbf{u}^2 + b\boldsymbol{\mathcal{K}}_\mathbf{u} + c \tag{27}$$

$$f(\boldsymbol{\mathcal{K}}_\mathbf{u} - 1) = S(\mathbf{u}, \boldsymbol{\mathcal{K}}_\mathbf{u} - 1) = a(\boldsymbol{\mathcal{K}}_\mathbf{u} - 1)^2 + b(\boldsymbol{\mathcal{K}}_\mathbf{u} - 1) + c \tag{28}$$

$$f(\boldsymbol{\mathcal{K}}_\mathbf{u} + 1) = S(\mathbf{u}, \boldsymbol{\mathcal{K}}_\mathbf{u} + 1) = a(\boldsymbol{\mathcal{K}}_\mathbf{u} + 1)^2 + b(\boldsymbol{\mathcal{K}}_\mathbf{u} + 1) + c \tag{29}$$

$$\boldsymbol{\mathcal{K}}^*_\mathbf{u} = \boldsymbol{\mathcal{K}}_\mathbf{u} - \frac{f(\boldsymbol{\mathcal{K}}_\mathbf{u} + 1) - f(\boldsymbol{\mathcal{K}}_\mathbf{u} - 1)}{2\left(f(\boldsymbol{\mathcal{K}}_\mathbf{u} - 1) - f(\boldsymbol{\mathcal{K}}_\mathbf{u}) + f(\boldsymbol{\mathcal{K}}_\mathbf{u} + 1) - f(\boldsymbol{\mathcal{K}}_\mathbf{u})\right)}, \tag{30}$$

where a, b, c are the coefficients of the parabola $f(\cdot)$, $\boldsymbol{\mathcal{K}}_\mathbf{u}$ is the plane index, which reaches the lowest cost at pixel $\mathbf{u}$, and $\boldsymbol{\mathcal{K}}^*_\mathbf{u}$ is the interpolated plane index at the extreme point. Thus the estimated depth of pixel $\mathbf{u}$ can be calculated by $d_\mathbf{u} = \frac{1}{\boldsymbol{\mathcal{K}}^*_\mathbf{u} \cdot \lambda_{min}}$.

Two criteria are used to remove the outliers. First, for the pixel with some sampling of the inverse depth, we cannot compute the similarity cost since the corresponding pixel projected in the measurement frame at that inverse depth is unknown. For those pixels $\mathbf{u}$, we mark it as undefined in the 3D cost volume at the corresponding inverse depth and later set $L_\mathbf{r}(\mathbf{u}, \boldsymbol{\mathcal{K}}_\mathbf{u})$ as zero in the SGM step to remove the influence of the missing cost. And finally, those pixels at the output depth image are set as undefined. Secondly, we drop the depth if it is not the absolute minimum of the fitting parabola. Specifically, the minimal cost located at the 0th or the $(L-1)$th plane must be ignored; in this case, the pixel's depth will be set as undefined. Also, if the sum of the cost of two neighbor planes is smaller than twice the minimal cost, the corresponding pixel will be set as undefined as well.

6.3 | Global dense mapping

We fuse all depth images obtained at different camera poses into a global dense map using an uncertainty-aware truncated signed distance field (TSDF) fusion approach. Our method is developed from the open source CHISEL TSDF implementation.[50] Improvements include uncertainty-aware depth fusion (Section 6.3.2) and algorithm parallelization (Section 6.3.3).

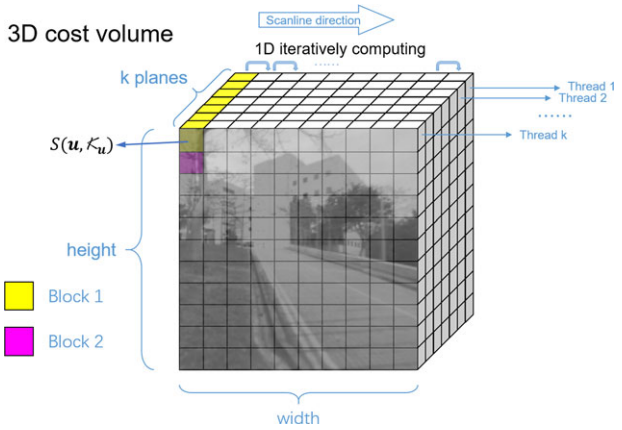


FIGURE 8 The semiglobal approximation acts as a 1D optimization problem that can be defined recursively as Eq. (25). We implement a parallel dynamic programming method on GPU by using CUDA. For each direction $\mathbf{r}$, we separate the recursion start point into different blocks. Each block has k threads enumerating k different depths, and all k threads run in parallel

6.3.1 | Truncated signed distance function (TSDF)

The whole environment can be modeled as a 3D volumetric signed distance field. The signed distance function $\phi(x) : \mathbb{R}^3 \to \mathbb{R}$ is the distance from voxel x to the its nearest surface, signed positive if the voxel is between the surface and the camera, and negative otherwise. Since we are only interested in reconstructing the surface, we use the truncated signed distance field:

$$\phi_\tau(x) = \begin{cases} \phi(x) & \text{if } |\phi(x)| \le \tau \\ \text{undefined} & \text{otherwise,} \end{cases} \tag{31}$$

where $\tau \in \mathbb{R}$ is the truncation distance.

Besides the truncated signed distance field, we store two more values at each 3D voxel. $C(x) : \mathbb{R}^3 \to \mathbb{R}$ is the photometric intensity, which is maintained like $\phi_\tau(x)$. $W(x) : \mathbb{R}^3 \to \mathbb{R}$ is the weight representing the confidence among measurements.

To achieve better memory efficiency, CHISEL uses a two-level hybrid data structure. The first level chunk is a spatially hashed map, which consists of a fixed grid of N_v^3 voxels. A chunk is allocated dynamically from a growing heap and indexed in a spatial 3D hash map based on their integer coordinates. The second level is the 3D cube voxel with size $\mathcal{L} \times \mathcal{L} \times \mathcal{L}$, where $\mathcal{L}$ is the mapping resolution in a metric. The sketch is shown in Figure 9.

Two depth scan fusion schemes are provided in CHISEL. Since we use the virtual pinhole camera in depth image estimation, projection mapping based on the camera frustum is more efficient than raycasting. The camera frustum is an axis-aligned bounding box that is dependent on the camera intrinsically and the nearest plane and farthest plane we tend to fuse. We first check each chunk that intersected with the camera frustum, and we allocate the chunk if it is the first time being checked. Then, every voxel in the chunks will be enumerated in the update procedure.

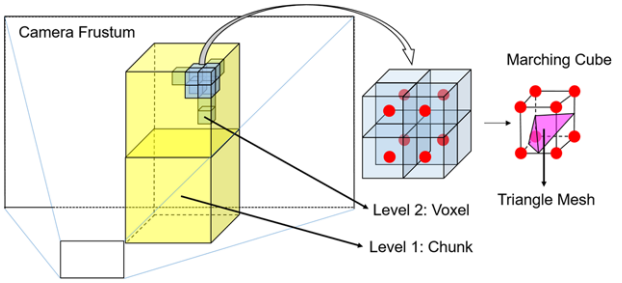


FIGURE 9 A general view of TSDF. Chunks that are intersected with the camera frustum are marked to be updated. For each marked chunk, voxels are enumerated and projected into the image plane for fusion in parallel. We apply the Marching Cube algorithm[51] when offline map visualization is needed

In the projection mapping method, each voxel would be projected onto the camera planes and compared with the depth value from the depth image. We denote the difference as $e_x \in \mathbb{R}^+$; only if the difference within the truncation distance $[-\tau, \tau]$, the voxel would be updated, otherwise it would be ignored . The voxel update algorithm is detailed in Algorithm 2.

ALGORITHM 2 Voxel projection mapping

procedure VOXEL UPDATE
  for $v \in V$ **do**
    $z \leftarrow ||v - o||$
    $d \leftarrow \mathbf{DepthImage}(\pi(K, v))$
    $u \leftarrow d - z$
    $\tau \leftarrow \Psi(d)$ ▷ Dynamic truncation distance
    if $u \in [-\tau, \tau]$ **then**
      $\phi_\tau(v) \leftarrow \frac{W(v)\phi_\tau(v)+\alpha(u)u}{W(v)+\alpha(u)}$
      $C_\tau(v) \leftarrow \frac{W(v)C_\tau(v)+\alpha(u)u}{W(v)+\alpha(u)}$
      $W(v) \leftarrow W(v) + \alpha(u)$
    end if
    if $u \in [\tau + \epsilon, d]$ **then** ▷ Space carving
      $\phi_\tau(v) \leftarrow$ undefined
      $C_\tau(v) \leftarrow$ undefined
      $W(v) \leftarrow 0$
    end if
  end for
end procedure

6.3.2 | Uncertainty-aware fusion

Since long-range visual depth estimation usually comes with larger depth uncertainty, we encode the uncertainty by using the dynamic truncation distance $\Psi(d)$. Define σ_λ as the standard deviation of the depth measurement in the inverse depth λ. Then the standard deviation of the depth measurement in depth d can be approximated using linearization:

$$\sigma_d = \sqrt{\left(\frac{\partial d}{\partial \lambda}\right)^2 \sigma_\lambda^2}, \tag{32}$$

where σ_λ is set to be λ_{min}, which is the minimal value in the inverse depth field. Thus, the truncation function is

$$\Psi(d) = \sigma_d \cdot S_{truncation}, \tag{33}$$

where $S_{truncation}$ is some constant truncation scale. Consequently, the truncation distance is dynamic and it is able to increase as the depth observation becomes larger.

6.3.3 | Parallelization

Note that the whole TSDF fusion computation complexity is approximately equal to the voxel update complexity, which is proportional to the voxel count. The update procedure for each voxel is independent, which allows us to accelerate the process by multithreading. We also decouple the map updating and publishing processes in order to guarantee the availability of the most recent map for trajectory planning. The map update runs as soon as a depth image (which can be from either of the two ROIs) comes, while the map publisher outputs the map to the trajectory planning at a constant frequency of 10 Hz. We used a mutex lock to guarantee the shared memory would not be conflicted.

6.3.4 | Map rendering for visualization

Although a visually appealing map is not necessary for autonomous navigation, we still provide a map rendering module that runs offline for visualization purpose. The Marching Cube algorithm[51] is used for map rendering. For every eight voxels, we take their centers to form a cube, and the signed distance field (SDF) value of each center point is approximately equal to its voxel's SDF value. Consider the sign of each SDF value. The whole cube has 256 different possible constitutions, which means there are 256 different ways to separate a cube into several triangle meshes (see Figure 9). A sample reconstruction result is visualized in Figure 10.

(a) Mesh Map (b) Actual Scene

FIGURE 10 A reconstructed mesh map (up) and the corresponding scene. While the dense reconstruction is not very accurate, it is sufficiently dense and consistent for autonomous navigation. Note that while the navigational map is generated online using TSDF, the mesh visualization is rendered offline

7 | TRAJECTORY PLANNING

In this section, we will present our trajectory planning method for quadrotor autonomous navigation in unknown environments with only a limited perception range. We use a sampling-based method directly applied on the dense map built in Section 6, where a path with asymptotic optimality is searched on a rapidly exploring random graph (RRG). The path, which consists of line segments, is converted to an initial collision-free trajectory for the quadrotor to pass through. Then an optimization method is utilized to smooth the trajectory and at the same time promote the dynamical feasibility and guarantee the safety of the trajectory. Both the front-end path searching and the back-end trajectory optimization are done within a predefined time limitation to meet the real-time requirement for navigation. We validate our planning method by demonstrating indoor and outdoor autonomous flights.

7.1 | Related work

The most rigorous challenge of an autonomous navigation mission for a quadrotor is that the trajectory not only should be collision-free in cluttered environments but also that it should be smooth and dynamically feasible to improve the flight performance of the quadrotor. Besides this, since the perception range of our proposed mapping module is limited in a short range, the trajectory should be generated sequentially within a planning horizon until the vehicle reaches its target. In addition, the trajectory should be generated in real-time with minimum time and computation cost to ensure that the quadrotor travels smoothly in the configuration space.

For an autonomous quadrotor system, the trajectory planning problem can be decoupled as front-end path searching and back-end trajectory refinement. Sampling-based methods are widely used to search for a feasible path in high-dimensional configuration space. A rapidly exploring random tree (RRT) was developed by LaValle.[52] In this method, samples are drawn randomly from the configuration space and guide the tree to grow toward the target; however, standard RRT algorithms are not asymptotically optimal. Karaman and Frazzoli[53] summarized and developed sampling-based motion planning methods that have asymptotic optimality, including RRT*, probabilistic road maps* (PRM*), and a rapidly exploring random graph (RRG). RRG is an extension of the RRT algorithm, as it connects new samples not only to the nearest node but also to all other nodes within a nearby region.

Although the sampling-based method is good for finding a safe path, the path consisting of piecewise line segments is not suitable for a flying vehicle to track. Mellinger et al.[54] pioneered a minimum snap trajectory generation algorithm, where the trajectory is represented by piecewise polynomial functions, and the trajectory generation problem is formulated as a quadratic programming (QP) problem. Unconstrained QP with a closed-form solution was formulated in Ref. 60, where a method is introduced to convert the original constrained QP problem to an unconstrained QP problem and directly get the optimal solution, thus bypassing the procedure of solving the constrained QP. Both numerical stability and computational efficiency are improved in this way.

Our previous work[55] carved a flight corridor in the environments and formulated the trajectory generation problem as a convex program with hard constraints. However, our previous method had an inherent nature that all the free space in the flight corridor is treated equally, making the generated trajectories to be close to obstacles since energy saving is the only cost in the objective function, and we ignored all the gradient information. This drawback motivates us to utilize the gradient information in the environments to add a penalty based on the distance to obstacles, thus pushing the optimized trajectory away from obstacles. We argue that given a safe initial trajectory, with a properly designed optimization procedure, a local optimal trajectory that is smooth, safe, and at the same time dynamically feasible can be obtained.

In this section, we propose a trajectory generation framework utilizing the gradient information of the configuration space and efficiently generating safe trajectories online. Our method employs a RRG expanded with heuristic in 3D space. An initial safe trajectory consists of straight lines is searched on the graph, followed by an optimization method to smooth the trajectory and constrain the dynamical feasibility, while at the same time also guaranteeing the safety of the trajectory. In the optimization part, we follow Ref. 56 to directly optimize the coefficients of the piecewise polynomial trajectory using nonlinear optimization. In Oleynikova et al.'s method,[56] even with a random restart, the success rate for generating a safe trajectory is around 60%, which is not acceptable in real-world navigation, especially in cluttered environments. However in our framework, with an initial safe

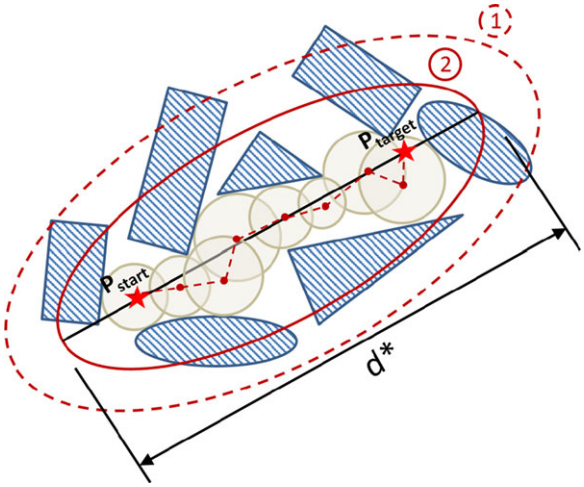


FIGURE 11 The heuristic sampling domain updated in path finding. The red dashed ellipsoid with marker 1 is the previous hyperellipsoid domain for generating samples. After a better solution d^* has been found, the sampling domain shrinks to the red solid ellipsoid with marker 2. Red stars indicate the start point and the target point for the path-finding mission. Gray spheres are safe regions found by sampling and nearest-neighbor query, and the red dashed polyline is the current best path searched by A^* in the graph

trajectory provided by the front-end path searching, we guarantee the success rate of generating a safe trajectory that is adaptive to complex environments.

7.2 | Sampling-based informed path searching

We adopt the method proposed in our previous work[55] to generate an initial path directly on unordered voxels output from the mapping module. Having the voxels that indicate obstacles in the environments, a rapidly exploring random graph is generated. Meanwhile a K-d tree built by the voxels is used to evaluate the safe volume of a given sampling point by a fast nearest-neighbor search. Nodes that have enough safe volume are added to the graph. After the generation and expansion of the graph, which consists of nodes with ball-shaped safe regions, A* is used to search a minimum distance path on it.

In this paper, we utilize the informed random sampling scheme that is introduced in Ref. 57 to improve the efficiency of the sampling as well as the optimality of the path. After the random graph reaches the target, which means the newly generated node contains the target location, we can determine that one path has been found, and we can obtain this path by back-tracking nodes in the graph. Having the distance of the path, we can use it to define a heuristic domain for subsequent sampling. We generate samples in a hyperellipsoid heuristic domain, with the start position and target position on its focal points. The transverse diameter of the hyperellipsoid is the minimum distance found so far, and the conjugate diameter of the hyperellipsoid is the straight line distance between the start and target position, as is shown in Figure 11. After enough samples are generated in the informed domain, A* would be used again to search for a better path and update the minimum distance as well as the heuristic domain. The procedure enjoys the asymptotic optimality that has been proven in Ref. 57. Thus if the random sampling and informed domain updating are done iteratively, the path will finally converge to the globally optimal path given infinite random samples.

For the sake of the real-time requirement in the autonomous navigation, the path-searching procedure should be terminated within a time limit. In our implementation, after finding the first feasible path, once the time exceeds the limit, no more samples will be generated and the best solution so far will be returned. Otherwise, A^* will be utilized to search a minimum distance path after a batch of samples, and the heuristic sampling domain will be updated based on the current best result. Having this anytime property, the path-searching module used in this paper adapts to the real-time requirement for the navigation mission. The output path will be initialized to an initial safe piecewise polynomial trajectory for the following optimization, with all the derivatives at waypoints set as 0, and time for each segment is allocated using an average velocity. Details about the formulation of the trajectory can be found in Section 7.3.1.

7.3 | Gradient-based trajectory optimization

7.3.1 | Problem formulation

Thanks to the differential flatness property of the quadrotor model, the full state space of the quadrotor system $\{x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, p, q, r\}$ is reduced to the 3D position and the yaw angle $\{x, y, z, \psi\}$ and their derivatives in which the trajectory is generated. A geometric tracking controller on $\mathbb{SE}(3)$[58] can be applied for feedback control. The trajectory consisting of piecewise polynomials is parametrized to the time variable t in each dimension μ out of x, y, z. The Nth-order M-segment trajectory of one dimension can be written as follows:

$$p_\mu(t) = \begin{cases} \sum_{j=0}^{N} \eta_{1j}(t-\mathcal{T}_0)^j & \mathcal{T}_0 \le t \le \mathcal{T}_1 \\ \sum_{j=0}^{N} \eta_{2j}(t-\mathcal{T}_1)^j & \mathcal{T}_1 \le t \le \mathcal{T}_2 \\ \vdots & \vdots \\ \sum_{j=0}^{N} \eta_{Mj}(t-\mathcal{T}_{M-1})^j & \mathcal{T}_{M-1} \le t \le \mathcal{T}_M, \end{cases} \tag{34}$$

where η_{ij} is the jth-order polynomial coefficient of the ith segment of the trajectory. $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_M$ are the end time of each segment, with a total time of $\mathcal{T} = \mathcal{T}_M - \mathcal{T}_0$.

Instead of using methods like CHOMP[59] to optimize the discrete waypoints and then generating the trajectory base on the waypoints,[10] we follow Ref. 56 to optimize directly on the coefficients of the piecewise polynomial trajectory. We also add a term that enforces high-order dynamic feasibility into the objective function. The formulation of the complete objective function can be written as

$$\min \quad f = w_1 f_s + w_2 f_c + w_3 (f_v + f_a), \tag{35}$$

where f_s is the regularized smoothness term to keep the trajectory smooth for the quadrotor to track, f_c is the cost of the clearance of the trajectory, and f_v and f_a are the penalties of the velocity and acceleration exceeding the dynamical feasibility. w_1, w_2, and w_3 are weight parameters to trade off the smooth collision clearance and dynamical feasibility.

The trajectory smooth cost is written as the integral of the square of the k_ϕ^{th} derivative:

$$f_s = \sum_{\mu\in\{x,y,z\}} \int_0^{\mathcal{T}} \left(\frac{d^{k_\phi} p_\mu(t)}{dt^{k_\phi}} \right)^2 dt, \tag{36}$$

In this paper, we minimize the fourth derivative of the position (snap) of the quadrotor to obtain a smooth trajectory, which is good for our vision-based state estimation and mapping system. Therefore k_ϕ is 4, and the order N of the piecewise polynomial we use is 8.

The objective function f_s can be written in a quadratic formulation $\boldsymbol{\eta}^T \mathbf{Q}_o \boldsymbol{\eta}$, where $\boldsymbol{\eta}$ is a vector containing all polynomial coefficients (η_{ij}) in all three dimensions of x,y,z, and $\mathbf{Q}_o$ is a Hessian matrix. We adopt the method proposed in Ref. 60 to obtain an unconstrained QP program for following optimization. We use a mapping matrix $\mathbf{M}$ to map the original variables—the polynomial coefficients—to the derivatives at each segment point of the piecewise polynomials:

$$\mathbf{M}\boldsymbol{\eta} = \mathbf{d}, \tag{37}$$

where $\mathbf{d}$ is the vector of all derivatives at the start and the end of each segment. And we use a selection matrix $\mathbf{C}$ to select and reorder the full derivatives $\mathbf{d}$ as $\mathbf{d}_F$ and $\mathbf{d}_P$, where $\mathbf{d}_F$ contains predefined fixed derivatives before the trajectory optimization, and $\mathbf{d}_P$ have free derivatives that are the variables to be optimized. Then we have

$$\mathbf{d} = \mathbf{C}^T \begin{bmatrix} \mathbf{d_F} \\ \mathbf{d}_P \end{bmatrix}, \tag{38}$$

$$\boldsymbol{\eta} = \mathbf{M}^{-1}\mathbf{C}^T \begin{bmatrix} \mathbf{d_F} \\ \mathbf{d}_P \end{bmatrix}. \tag{39}$$

The cost function of the smoothness term can be written as

$$f_s = \begin{bmatrix} \mathbf{d_F} \\ \mathbf{d}_P \end{bmatrix}^T \mathbf{C}\mathbf{M}^{-T}\mathbf{Q}_o\mathbf{M}^{-1}\mathbf{C}^T \begin{bmatrix} \mathbf{d_F} \\ \mathbf{d}_P \end{bmatrix}. \tag{40}$$

Denote $\mathbf{C}\mathbf{M}^{-T}\mathbf{Q}_o\mathbf{M}^{-1}\mathbf{C}^T$ as matrix $\mathbf{U}$. Then the cost function can be written in a partitioned form as

$$f_s = \begin{bmatrix} \mathbf{d_F} \\ \mathbf{d}_P \end{bmatrix}^T \begin{bmatrix} \mathbf{U_{FF}} & \mathbf{U}_{FP} \\ \mathbf{U}_{PF} & \mathbf{U}_{PP} \end{bmatrix} \begin{bmatrix} \mathbf{d_F} \\ \mathbf{d}_P \end{bmatrix}. \tag{41}$$

The Jacobian $\mathbf{J}_s$ of f_s with respect to the free variables $\mathbf{d}_P$ in the μ (μ out of x, y, z) axis can be computed as

$$\mathbf{J}_s = \left[\frac{\partial f_s}{\partial \mathbf{d}_{P_x}}, \frac{\partial f_s}{\partial \mathbf{d}_{P_y}}, \frac{\partial f_s}{\partial \mathbf{d}_{P_z}} \right],$$
$$\frac{\partial f_s}{\partial \mathbf{d}_{P_\mu}} = 2\mathbf{d}_F^T\mathbf{U}_{FP} + 2\mathbf{d}_P^T\mathbf{U}_{PP}. \tag{42}$$

For the cost of collision on the trajectory, we need a differentiable cost function to penalize the distance value, and we want the cost to rapidly grow to infinity near the obstacles and to be flat away from the obstacles. If we provide a proper collision-free initial trajectory, the collision cost near the obstacles will dominate the objective function and prevent the trajectory from colliding with them. The cost function we select is an exponential function. At a position in the map with a distance value δ, the cost $c(\delta)$ is written as

$$c(\delta) = \rho \cdot \exp(-(\delta - \delta_0)/\nu), \tag{43}$$

where ρ is the magnitude of the cost function, δ_0 is the threshold where the cost starts to rapidly rise, and ν controls the rate of the function's rise. The cost is negatively correlated to the distance value. As illustrated in Ref. 59, formulating the collision cost as the line integral of the distance value over the arc length of the trajectory is more proper than a straightforward integration across time. And for numerical calculation, we discretize the integral and formulate it as a summation of costs on different time stamps:

$$f_c = \int_s c(p(t))ds = \int_{\mathcal{T}_0}^{\mathcal{T}_M} c(p(t))\|v(t)\|dt = \sum_{k=0}^{\mathcal{T}/\delta t} c(p(\mathcal{T}_k))\|v(\mathcal{T}_k)\|\delta t, \tag{44}$$

where $\mathcal{T}_k = T_0 + k\delta t$, $p(\cdot)$ and $v(\cdot)$ are the position and velocity of a point along the trajectory in the body frame to world frame, respectively.

The Jacobian $\mathbf{J}_c$ of f_c with respect to the free variables $\mathbf{d}_{P_\mu}$ can be obtained as follows:

$$\mathbf{J}_c = \left[\frac{\partial f_c}{\partial \mathbf{d}_{P_x}}, \frac{\partial f_c}{\partial \mathbf{d}_{P_y}}, \frac{\partial f_c}{\partial \mathbf{d}_{P_z}} \right],$$
$$\frac{\partial f_c}{\partial \mathbf{d}_{P_\mu}} = \sum_{k=0}^{\mathcal{T}/\delta t} \left\{ \nabla_\mu c(p(\mathcal{T}_k))\|v(\mathcal{T}_k)\|\boldsymbol{\mathcal{T}}\boldsymbol{\Lambda}_{p\mu} + c(p(\mathcal{T}_k))\frac{v_\mu(\mathcal{T}_k)}{\|v(\mathcal{T}_k)\|}\boldsymbol{\mathcal{T}}\mathbf{V}_m\boldsymbol{\Lambda}_{p\mu} \right\} \delta t, \tag{45}$$

where $\boldsymbol{\Lambda} = \mathbf{M}^{-1}\mathbf{C}^T$ and $\boldsymbol{\Lambda}_{p\mu}$ is the right block of matrix $\boldsymbol{\Lambda}$, which corresponds to the free derivatives $\mathbf{d}_{P_\mu}$ in the μ axis, and $\nabla_\mu c(p)$ is the gradient of the collision cost at p in the μ axis. $\mathbf{V}_m$ maps the polynomial coefficients of position to polynomial coefficients of velocity. At time $\mathcal{T}_k$, $\boldsymbol{\mathcal{T}} = [\mathcal{T}_k^0, \mathcal{T}_k^1, \ldots, \mathcal{T}_k^n]$.

For formulating the cost on the dynamical feasibility of velocity, we generate an artificial cost field on velocity between the maximum velocity and minus the maximum velocity in the x, y, and z axes. And we write f_v as the sum of the line integral of the velocity in the x, y, and z axes. The cost function of the high-order dynamical feasibility is also an exponential function, since it is good at penalizing when close to or beyond the limit of bounds and staying flat when away from the bounds. $c_v(v)$ is the cost function applied on the velocity, which has the same form as in Eq. (43). f_v is written as follows, where a_μ is the acceleration:

$$f_v = \sum_{\mu\in\{x,y,z\}} \int_s c_v(v_\mu(t))ds = \sum_{\mu\in\{x,y,z\}} \int_{\mathcal{T}_0}^{\mathcal{T}_M} c_v(v_\mu(t))\|a_\mu(t)\|dt. \tag{46}$$

The Jacobian $\mathbf{J}_v$ of f_v has a similar form to Eq. (45). We omit the formulation of the acceleration feasibility cost f_a for brevity.

Having the total Jacobian $\mathbf{J} = w_1\mathbf{J}_s + w_2\mathbf{J}_o + w_3(\mathbf{J}_v + \mathbf{J}_a)$, general nonlinear optimization methods can be utilized to minimize the objective. In the optimization, all derivatives at middle waypoints, including the position, velocity, and acceleration, are free variables in $\mathbf{d}_P$. The derivatives at the start and target points are fixed variables in $\mathbf{d}_F$.

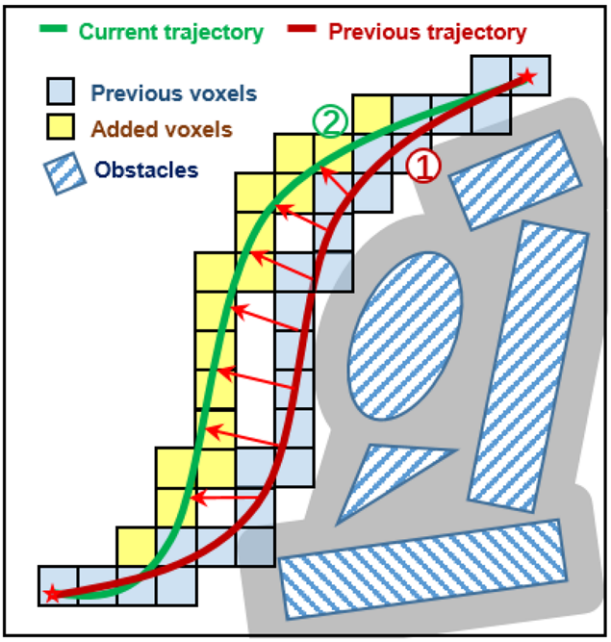


FIGURE 12 Illustration of the local voxel hashing. Shapes with shades are obstacles. The red and green curves are trajectories before and after one iteration, respectively. Blue voxels have already been recorded, while yellow voxels are newly added. Gray areas are collision margins that are near obstacles but are still considered safe. Red arrows roughly show the gradient direction

7.3.2 | Local voxel hashing for collision cost evaluation

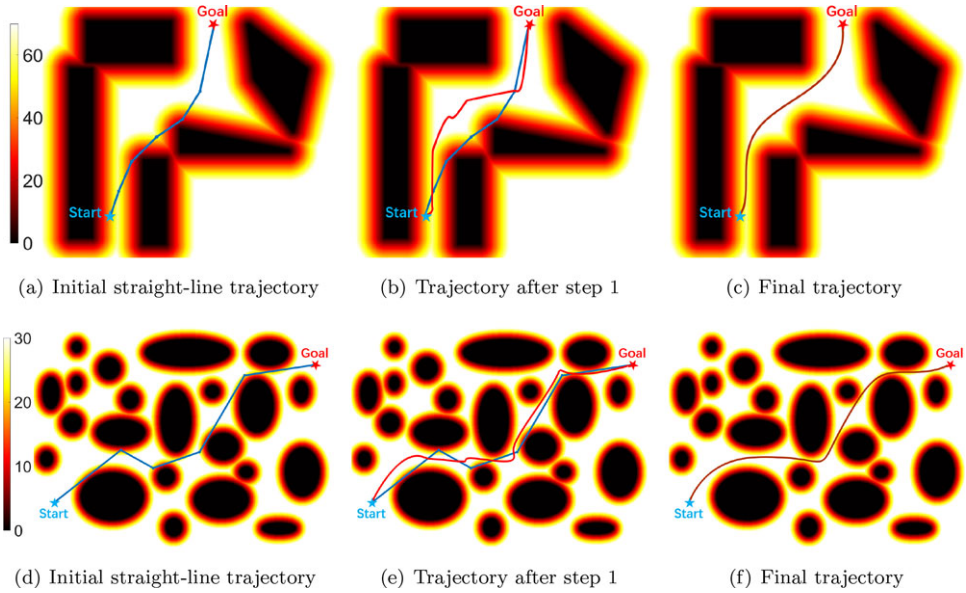
To evaluate the cost of the collision penalty and get the gradient information along the trajectory, the most commonly used method is to maintain a distance map. But even for a local map with a short range, the maintenance and updating of the complete distance field is costly. Furthermore, maintaining a distance field map and getting the gradient by taking the difference on the map in the x, y, and z directions only provides an approximated gradient and can be easily undifferentiable where the distance value is indeed not continuous. Thus we propose a new light-weight method to help us evaluate the collision cost.

Having the initial trajectory generated in Section 7.2, only a small part of the space will be evaluated during the optimization. Thanks to the sparsity of the distribution of the evaluated voxels in the map, we can accelerate the procedure by voxel hashing. We evaluate the cost and gradient of a point on the trajectory and simply hash the voxel it belongs to according to a predefined resolution (voxel size). For each point, we do a nearest-neighbor search using the K-d tree. Since the tree has been built in the path-searching module, there is no extra cost for building the tree. The nearest-neighbor query only has $O(\log N)$ complexity, where N is the number of voxels. In this way, the Euclidean distance in 3D space of a point to its nearest neighbor is obtained, and at the same time we get the position of the nearest neighbor, which directly indicates the direction of the gradient. During the optimization, we cache all voxels that have been queried with the cost and gradient information for accelerating further iterations, as is shown in Figure 12.

7.3.3 | Two-step optimization framework

Nonlinear optimization is applied to smooth the collision-free initial trajectory, and at the same time ensure safety and dynamical feasibility. The initial trajectory, which geometrically follows the collision-free path (see Section 7.2), is used as the initial value. We apply a two-step optimization strategy that can be summarized as follows:

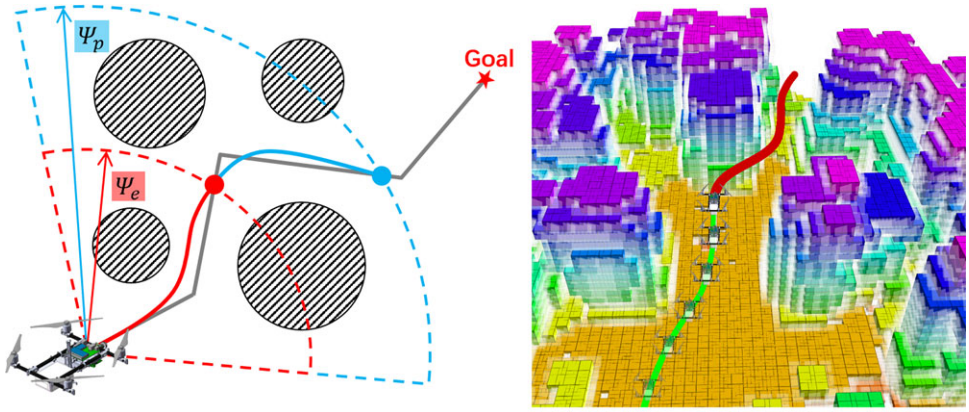
1. Optimize the objective with only the collision cost. All the free variables, including the positions of middle waypoints on the initial



(a) Initial straight-line trajectory (b) Trajectory after step 1 (c) Final trajectory

(d) Initial straight-line trajectory (e) Trajectory after step 1 (f) Final trajectory

FIGURE 13 Trajectory generated in a 2D sparse and dense map. The color code shows the value in the distance field used in the trajectory generation. The initial trajectory, shown as a blue curve, is a collision-free straight-line trajectory that is found by the path searching module. After the first step of the optimization, the trajectory and middle waypoints are pushed away from the obstacles, as is shown by the red curve in (b). The final trajectory is shown as a brown curve after the second step of optimization



(a) Planning horizon and execution horizon (b) Trajectory generated in the planning horizon

FIGURE 14 Illustration of the receding horizon incremental planning strategy. In (a), the global guiding path is shown with the gray line. The blue curve and red curve are the planning trajectory and the execution trajectory, respectively. See Section 7.4 for details. A trajectory generated in the planning horizon during the indoor flight is shown in (b). The color code shows the height of the obstacles; the newly generated trajectory is shown in red, while trajectories that have been executed are shown in green

trajectory, will be pushed to the basin of the distance field from the neighborhood of obstacles.

2. Reallocate the time and reparametrize the trajectory to time according to current waypoint positions. Then optimize the objective with a smoothness term and a dynamical penalty term added. The optimizer smooths the safe trajectories and squeezes the dynamic infeasibility, and at the same time prevents the trajectories from moving near the obstacles again.

Since the path finder may place waypoints close to obstacles, optimizing only on the collision cost in the first step can lead the trajectory to converge quickly to a much safer trajectory. After that, the reallocation of time in each segment of the trajectory according to the current waypoints' positions makes the time allocation more appropriate for following optimization. The entire optimization procedure finishes when the termination condition is met or the predefined time limit for optimization is reached. Results are shown in Figure 13. Although we have a nonconvex objective function, with a proper initial trajectory and a cost function that rapidly goes to infinity near the obstacles, a local minimum solution that is safe can be obtained by using general nonlinear optimization methods. Since in each term of our objective the cost function is at least twice differentiable, the Hessian of the objective can also be derived to obtain better convergence for optimization. We omit the formulation of the Hessian in this paper for brevity.

7.4 | Receding horizon planning scheme

Due to the limited sensing range of our mapping module, we use a receding horizon incremental planning strategy.[61] Before the flight starts, a target point is set and a global path finder is utilized to find a guiding path to reach the target. During the navigation, a local planner is used to find the path and optimize the trajectory within the planning horizon Ψ_p. The trajectory will only be executed in an execution horizon Ψ_e, after which the local planner will be called again, as is shown in Figure 14. And if the newly updated map has a collision with the trajectory that is going to be executed, the local planner will be used immediately to generate a new trajectory. The planning target for the local planner is selected first on the global guiding path, and if the target is occupied (e.g., if it has obstacles), the planner will automatically adjust the target and search for a free point in a nearby region. In the motion planning module, both the front-end path searching and the back-end trajectory optimization have predefined time limits, which control the termination conditions of the whole pipeline.

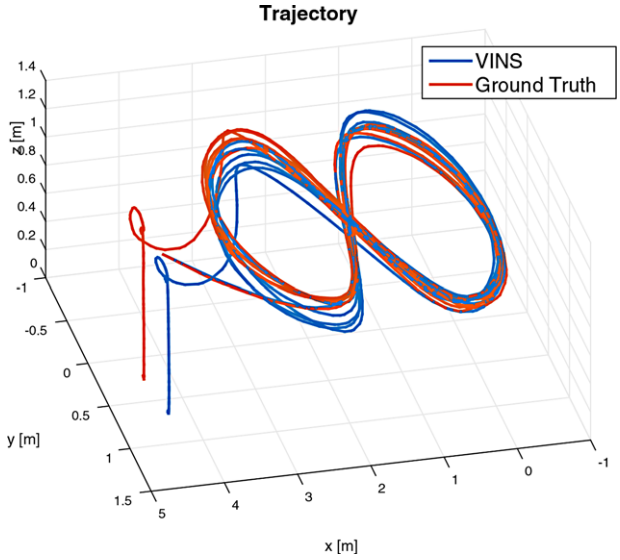


FIGURE 15 The trajectory of the aggressive motion. VINS outputs are used as real-time feedback in a closed control loop. VINS results are compared with OptiTrack. The total length is 75.3 m and the total drift is 0.35 m

The existence of the global guiding path guarantees that the quadrotor will finally reach the target, while during the navigation the local planner is used to avoid any possible collisions within the map range. Note that all unknown space (outside the perception range) is treated as free in the global path finder and occupied in the local planner. In a real implementation, when observations are not sufficient, an

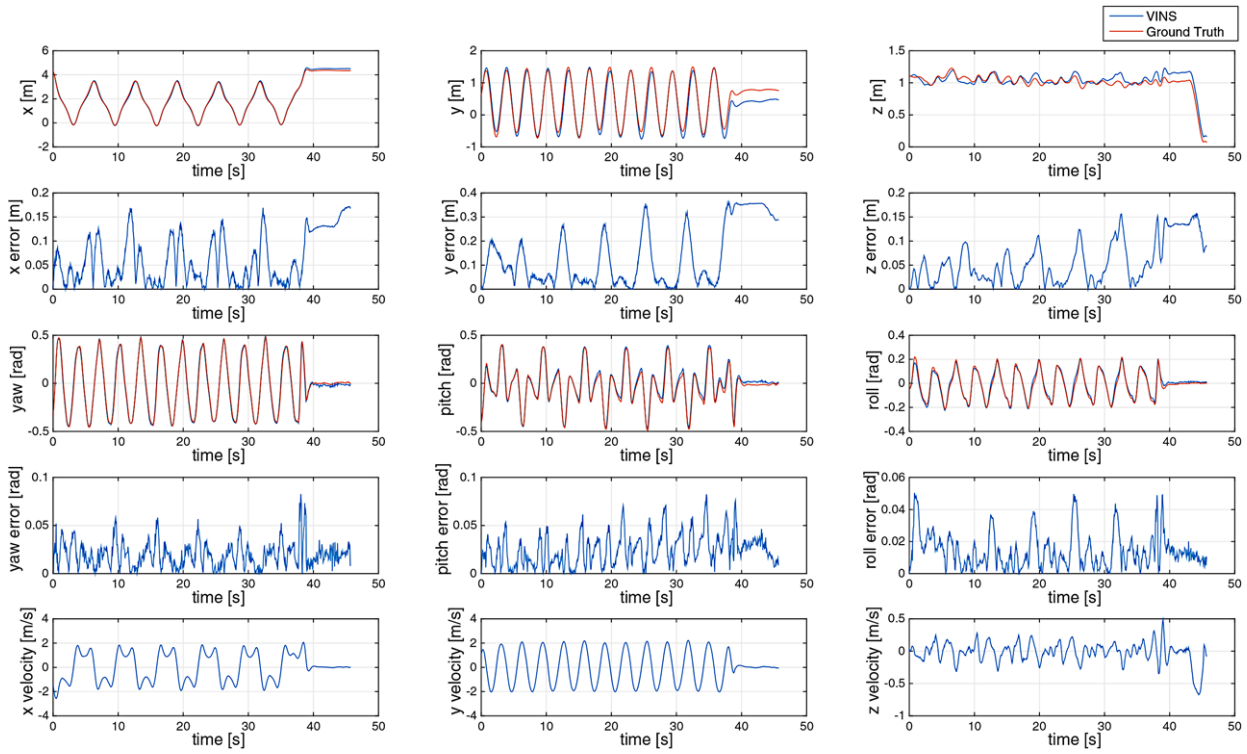


FIGURE 16 The top four rows are translation and orientation values and their corresponding error. The bottom row is the velocity plot

unknown area may block the way for finding a path by the local planner, especially when there is no significant base-line for motion stereo. In this situation, the quadrotor will generate safe but short trajectories in the free space, and it will explore the surrounding environments until the map is fully observed and a feasible path is found. Details about the autonomous flights and the exploration strategy are shown in the video.

8 | EXPERIMENTAL RESULTS

The experiments section is divided into four parts. The first two parts evaluate the performance of the main modules in our system, where the first experiment compares the state estimation with a motion capture system, namely OptiTrack* in aggressive motion. The second experiment evaluates the mapping result by using both synthetic and real data. The rest of the experiments test the whole autonomous flight system in both indoor and outdoor environments. Both trajectories and 3D reconstruction results are presented. More details can be found in our experiment videos.

8.1 | Feedback control using monocular VINS

In this experiment, we test the performance of autonomous trajectory tracking with VINS in aggressive motion. The quadrotor is commanded to track a figure-eight pattern with each circle being 1.0 m in radius, as shown in Figure 15. The quadrotor follows this trajectory six times continuously during experiment. The 100 Hz state estimation results achieve the real-time feedback to control the quadrotor to follow the predesigned aggressive trajectory, as we mentioned in Section 5.6.

The robustness and accuracy are of vital importance to this real-time onboard experiment. The linear velocity reaches 2.3 m/s in this experiment, and the maximum tails of the quadrotor are more than 25 degrees during flying. The final drift is 0.35 m, relative to the total 75.3 m path length, by comparing our proposed real-time estimation with OptiTrack. The final percentage of drift is 0.46%. The details of the translation and rotation as well as their corresponding error are shown in Figure 16.

8.2 | Dense depth estimation

Instead of directly evaluating the precision of the dense 3D reconstruction, we compare the depth images that mainly affect the performance of the reconstruction. Two experiments on synthetic data and realistic data, respectively, are performed to evaluate our algorithm qualitatively.

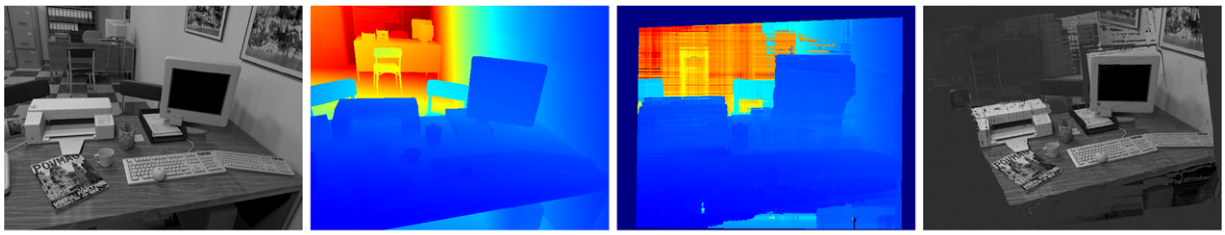
8.2.1 | Synthetic experiment

In this part, we evaluate our motion stereo depth estimator by using a public synthetic Variable Frame-Rate Imperial College (VaFRIC)[62] dataset.† This dataset contains a synthetic RGB image sequence together with a depth image sequence rendered from POVRay‡ with a resolution of 640 × 480. Also, the ground truth of the camera poses

* http://optitrack.com/

† http://www.doc.ic.ac.uk/~ahanda/VaFRIC/index.html

‡ http://www.povray.org/



(a) Grayscale image (b) Ground truth depth (c) Our depth (d) Our pointcloud

FIGURE 17 Parts (a) and (b) are the grayscale image and the ground truth depth image from the VaFRIC dataset, respectively. Part (c) visualizes the depth image generated from our method. Blue means the pixel is close to the image plane, while red represents the pixel with large depth. In particular, deep blue around the border indicates the nonoverlap region, which has undefined depth. In addition, the corresponding point cloud backprojected from our depth image is shown in (d) to provide an intuitive view of our depth accuracy

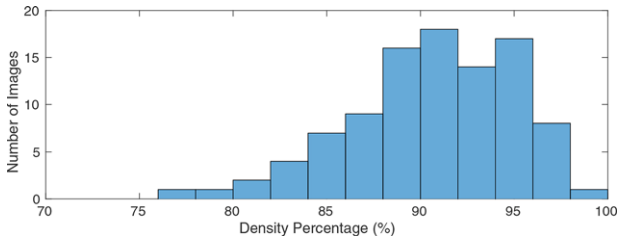


FIGURE 18 Density percentage: We select a sequence of images (98 images) for experiments. The number of images with different depth density is counted and plotted in the histogram. Most of the depth images keep the density percentage higher than 88%

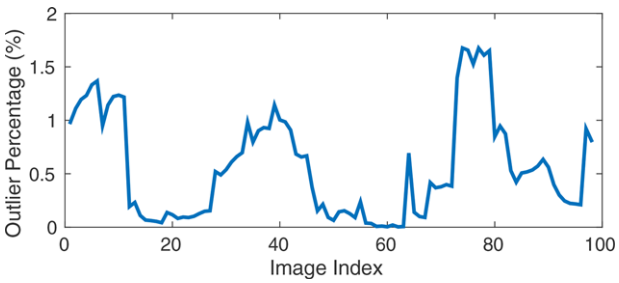


FIGURE 20 Outlier ratio curve. The pixel with a depth error larger than 1% is considered as an outlier

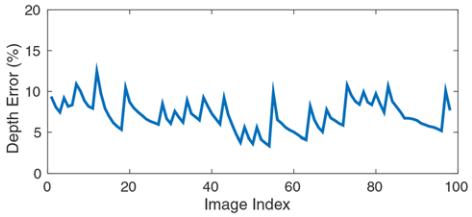
is given. By using this ground truth as the state estimation input, in each image we compare the depth we optimized with the ground truth depth. Three different quantitative indicators are shown: depth density, average depth error, and outlier ratio.

As illustrated in Figure 17, a sample of grayscale image, ground truth depth image, our depth image, and the corresponding point cloud are shown for an intuitive visualization. As we mentioned in Section 6.2.4, two outlier rejection criteria cause the depth image not to be fully dense. Although the density is lowered, it is still double or triple that in semidense mapping, and it is sufficient to provide full awareness of the environments. Figure 18 presents the density ratio during the experiment.

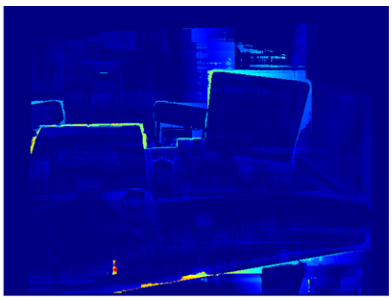
We compare the average absolute value of the depth error and plot the average pixel depth precision curve in Figure 19(a). Since most of the pixel depths coincide with the ground truth in an acceptable error, the depth error image is also shown [see Figure 19(b)] for visualizing the distribution of the error. The pixel whose depth error against the ground truth is larger than 1% is considered as the outlier, and the outlier ratio curve is illustrated in Figure 20.

8.2.2 | Real-world experiment

In this experiment, we choose our indoor laboratory environment as the experiment area. We do a comparison with another similar work, REMODE,[5] which is an open-source real-time dense reconstruction implementation. Both 10 Hz VINS pose estimation results and 10 Hz left cropped images are collected in the flight. After data collection, we

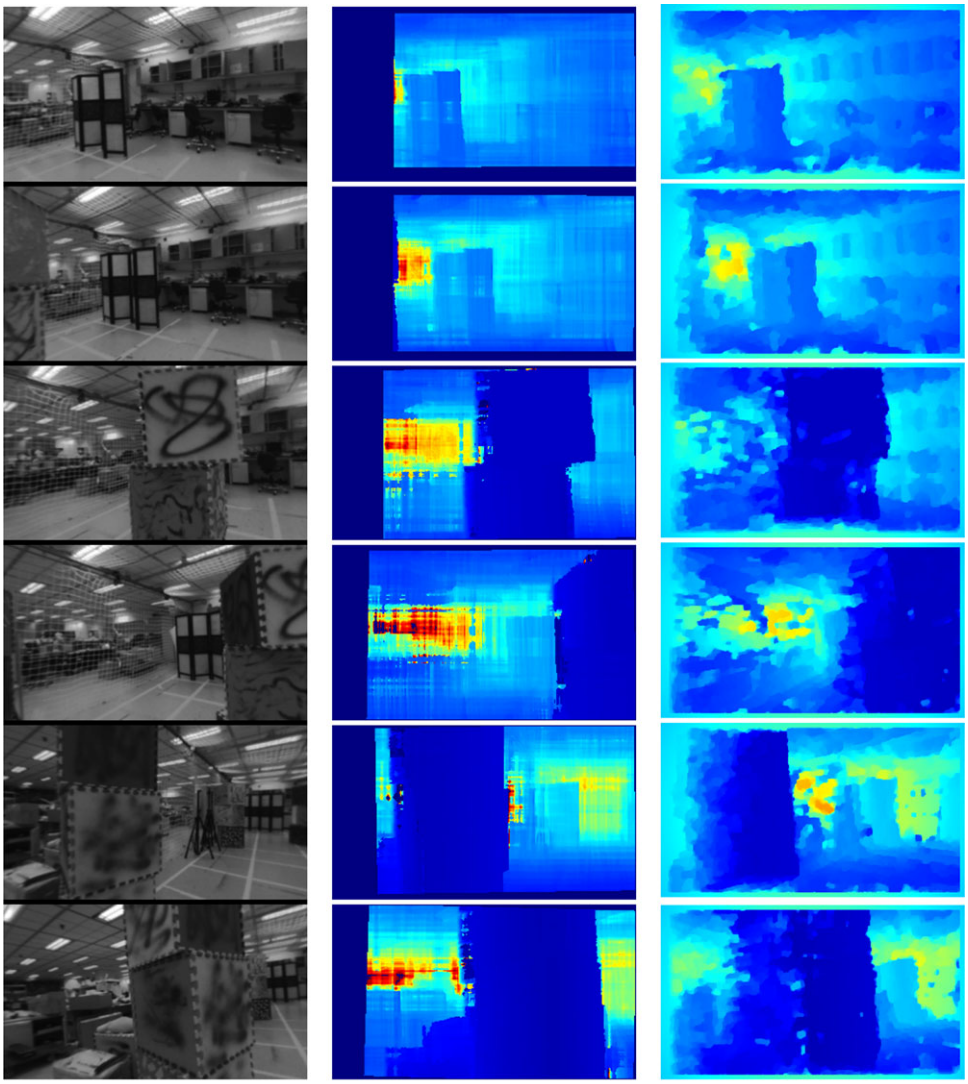


(a) Average depth error: The periodic pattern in depth error is a result of keyframe switching. Each time a keyframe is changed, the set of measurement frames needs to be re-aggregated. As a result of using fewer measurement frames for depth estimation, the error is larger. As more measurement frames come in, the error decreases.



(b) Depth error image: The light color indicates the high depth error area. Suffering from the textureless influence, the contour between the monitor and the wall gets wrong estimation. Also, occlusion is another issue for high depth uncertainty in the object contour.

FIGURE 19 Average depth error curve and a sample depth error image



(a) Grayscale images (b) Our depth images (c) REMODE depth images

FIGURE 21 Sampled results in indoor environments. The first column is the grayscale images with a resolution of 320 × 184. The second column is our depth image result sequence, where blue indicates the small depth region and red represents the pixel with large depth. The third column is the corresponding depth image sequence generated by REMODE

process our depth estimator and REMODE offline on the desktop computer. To maintain consistency, our depth estimator and REMODE both use the same pose estimation results solved from our VINS. Since there is no depth ground truth, we compare the corresponding depth image outputs and illustrate them in Figure 21. The results show that even in a textureless region or in low-resolution images (320 × 184), which give less detailed information about the scene, our semiglobal optimization method can get a reliable depth image.

TABLE 2 Approximated timing statistics

Module	Time Consumption (ms)	Frequency (Hz)
VINS–Front-End	20	20
VINS–Back-End	54	10
Depth Map Optimizer	95	10
TSDF Fusion	36	10

8.3 | Indoor experiment

We validate our system in both indoor and outdoor environments. Considering the lower mapping resolution requirement in the motion planning module, we set our voxel map resolution as 0.15 m for high efficiency. The average time consumption is measured in Table 2.

We design two challenging indoor autonomous flight experiments in the laboratory environments. In the experiment, our mapping module provides *a*5 m range map with 0.15 m voxel resolution, and the local planning horizon is set as 5 m and the execution horizon is 2 m. Computing time limits for path finding and trajectory optimization are set as 15 and 25 ms, respectively.

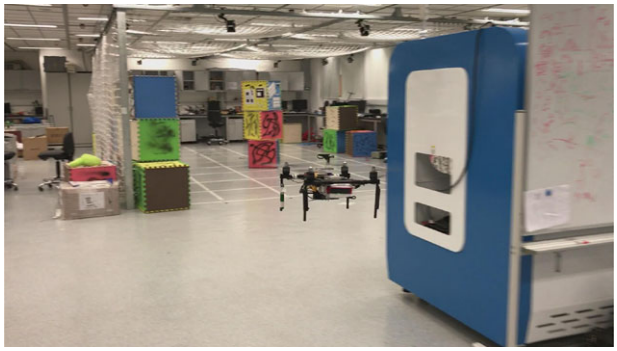


FIGURE 22 We take our indoor experiments in our laboratory, which has a size of 16 m × 7 m. The obstacles we position on the ground are randomly deployed. Our quadrotor takes off in the front of the white board and flies through the optimized trajectories to the destination of the navigation

8.3.1 | Indoor fast obstacle avoidance

To validate that our system is able to handle small-scale indoor autonomous navigation, we put two obstacles sparsely in the flight area (Figure 22), and the quadrotor is expected to quickly pass by the obstacles to the destination with full autonomy. The quadrotor first launches the localization module and takes off. After setting a destination point behind the obstacle, the quadrotor starts to fly forward autonomously without any prior knowledge about the environments. The surrounding environments are reconstructed in real time, and the voxel map is fed into the trajectory planning module, where safe and smooth trajectories that avoid possible collision with the environments are generated. The quadrotor follows the optimized trajectory in a receding horizon incremental planning scheme, and new trajectories are generated in a planning horizon and executed in an execution horizon. And if the newly updated map has a collision with the trajectory that is going to be executed, the planner will be used immediately to generate a new safe trajectory. Details about the trajectory planning can be found in Section 7. A series of snapshots of the experimental result are visualized in Figure 23. The corresponding mesh map is reconstructed offline at a higher resolution, which emphasizes our accuracy of localization and mapping.

The average speed during this experiment is around 1.1 m/s, while the highest velocity is about 2.2 m/s. More details about the quadrotor's linear velocity are plotted in Figure 24. For robust application, the most critical property we consider is the system delay. As we mentioned in Section 7, the planning module runs in an anytime style where we usually limit it in 40 ms. The whole system delay depends on our mapping part. The practical time cost measurements for all modules related with the mapping part are plotted in Figure 25. The overall average delay is around 230 ms, which is enough for our demonstration of autonomous navigation but still can be improved. For the sake of reaching a high pose precision of dense 3D reconstruction, in our experiments we use the 10 Hz VINS pose estimation results for mapping instead of the 100 Hz estimated poses from IMU propagation, which will incur less delay. Therefore, we are able to reduce the delay to around 150 ms for an extremely high speed mission.

More details about the indoor fast obstacle avoidance experiments can be found in our video (http://www.ece.ust.hk/~eeshaojie/jfr2017yi.mp4).

8.3.2 | Indoor complex environment navigation

In the second experiment, we verify our system performance in localization, mapping, and planning in more complex environments. Dozens of experiments have been demonstrated to show the robustness of our system.

In experiments, we put more obstacles on the ground to build a complex and narrow environment (see Figure 26). With the same starting point and the same destination point, the second indoor experiments are more challenging since most of the feasible space is occluded by obstacles, which makes our quadrotor act like it is exploring. Thanks to our fisheye camera, our system has a large FOV to detect obstacles, even if obstacles are parallel with the quadrotor.

We capture some final reconstructed scene results with optimized trajectories, and we show them in Figure 27. The accuracy and consistency of our reconstructed maps are shown.

8.4 | Autonomous flight in outdoor environments

In this experiment, we choose a forest-like area on a sloped hillside to validate our autonomous flight system. This evidences that our system is able to operate in full 3D environments with various shapes of obstacles. Figure 28 shows the environment at the start point. The start point is at the foot of the hillside. Our quadrotor detects and avoids the thin tree branches and follows the trajectory that is generated in real-time from the planning module and finally hovers at the destination point. An offline rendered mesh map is presented in Figure 29.

By using autoexposure, our system will not be affected by the illumination difference between indoor and outdoor environments, which is a key advantage compared to RGB-D-based approaches. The depth image results in the outdoor environment are shown in Figure 30.

More experimental trials and online visualization can be found in the video attachment.

9 | DISCUSSIONS AND CONCLUSION

9.1 | Lessons learned

9.1.1 | System development

The software architecture of our system has a potential ability to be set up on one NVIDIA TX1 only. However, we choose to assemble a heterogeneous computing hardware. This provides redundant computation power, and it benefits rapid algorithm development. The communication between the two computing devices is carried out by ROS. As we mentioned in Section 3.2, each module is implemented into a ROS node, which reduces debugging time during system integration. Nevertheless, load balance is one of the aspects we have not optimized since each node must be exactly loaded on one of these two devices,

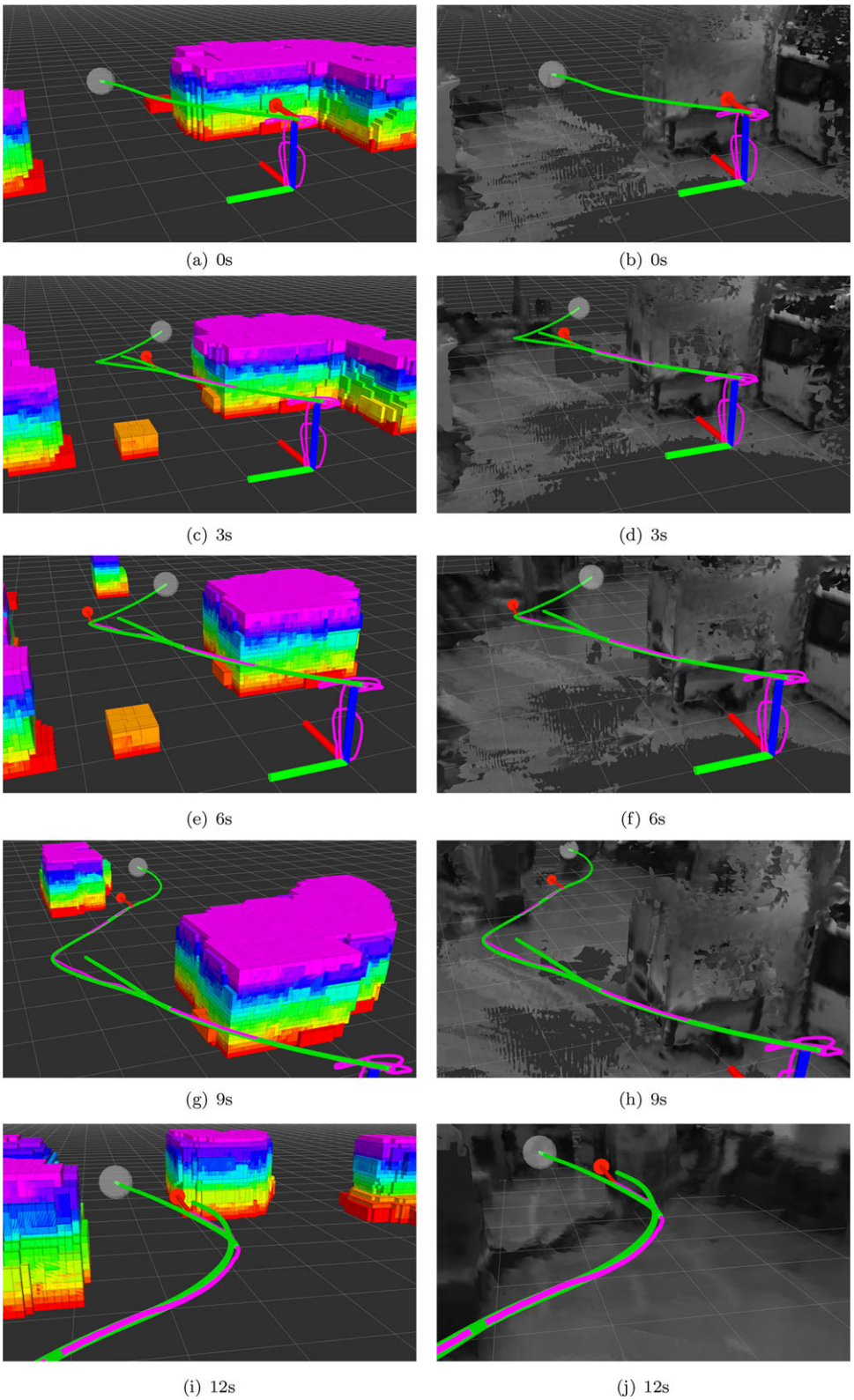
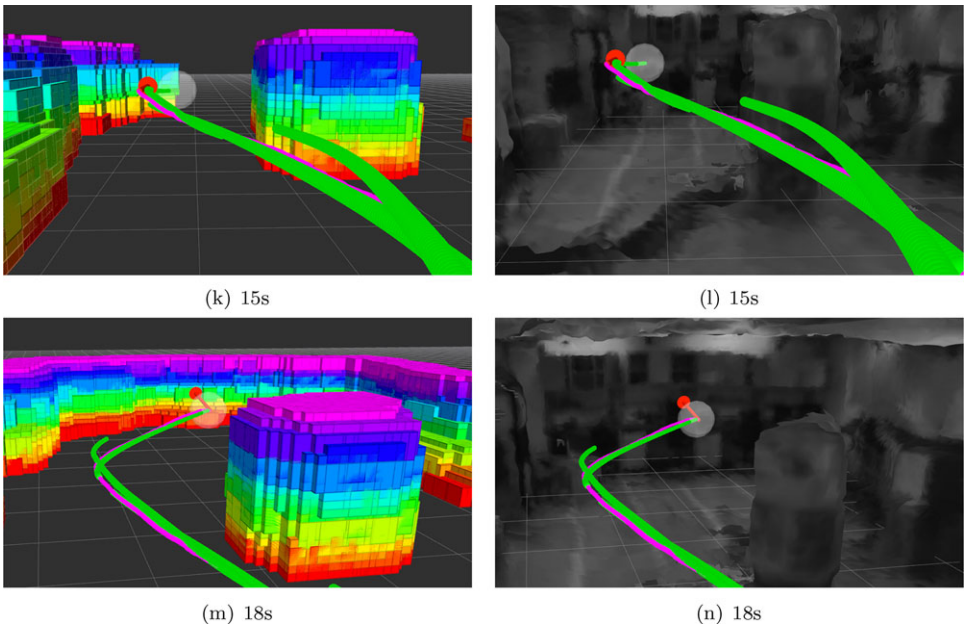


FIGURE 23 We snapshot every 3 s of the map and the trajectories during the flying mission. The first column shows the voxel map, which is fed into the planning module. For safety consideration, every occupied voxel has been inflated by 0.15 m. The second column is the corresponding mesh map. The red arrow represents the current position of the quadrotor. The transparent white ball indicates the local trajectory destination, and the green lines are the optimized trajectories. Purple curves show the actual path



(k) 15s (l) 15s

(m) 18s (n) 18s

FIGURE 23 Continued

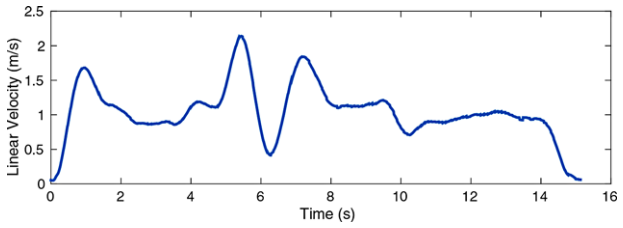
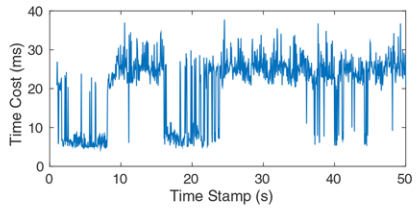


FIGURE 24 The linear velocity of our quadrotor during the planning mission. The overall average linear velocity is around 1.1 m/s while the highest speed is about 2.2 m/s

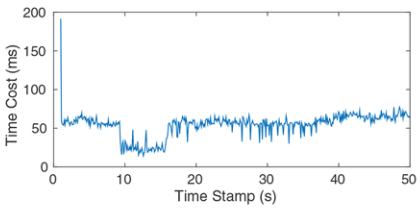
which raises a problem about optimal task allocation because of different computing resource requirements for different nodes.

9.1.2 | Experiments

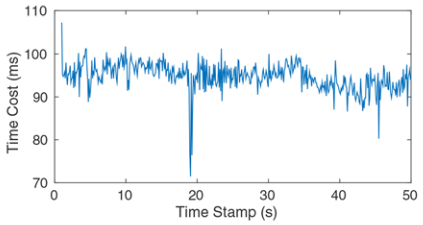
We choose a set of parameters to achieve the best tradeoff between mapping quality and processing speed, following two rules: 1) minimum resolution in TSDF to satisfy planning requirements; 2) maximum image ROI and maximum number of sweeping planes that maintain real-time processing and appropriate mapping range for rapid planning.



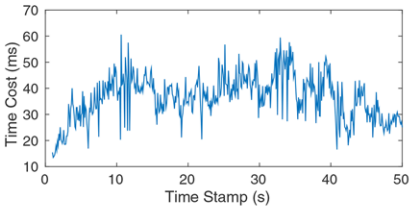
(a) VINS - Front-End: VINS front-end runs in 20 Hz for robust feature tracking. The overall average time cost is around 20.48 ms.



(b) VINS - Back-End: The average consuming time is about 54.15 ms except the initialization period.



(c) Depth Map Optimizer: Two motion stereo depth optimization nodes process two ROI images parallel and respectively. The total average processing time is around 94.72 ms.



(d) TSDF Fusion: TSDF costs about 36.37 ms for signed distance updating together with occupied grid map rendering.

FIGURE 25 The running time of four major modules measured during the experiments

FIGURE 26 We randomly position seven obstacles in the flight area. After setting a destination near the forest border of the flight area, the quadrotor flies through the feasible space while avoiding any obstacles

FIGURE 28 In our outdoor experiment, we set our start point at the foot of the slope, as the figure shows. After takeoff, the quadrotor autonomously flies through the forest. The trajectories are generated in a receding horizon strategy, and once the trees are detected as obstacles that block the path, the replan is immediately triggered and the quadrotor follows the new path to avoid the trees

The perception range plays a tradeoff role. Larger maps help the planning module to control the robot to fly at a higher speed, but they result in more delay due to computation requirements. This reduces the flight speed due to safety reasons. Therefore, in our experiments, we only use the map within a 5 m range, even though the depth estimator is able to provide a longer perception range.

As we mentioned in Section 3.2.2, we use rviz on the airborne computer for visualization. However, rviz consumes almost all the computing resource when it renders all meshes. In addition, we found that the grid map is sufficient for online monitoring and debugging. Therefore, during online experiments, only grid maps are visualized.

9.1.3 | Failure cases

It is well known that the monocular visual-inertial system will suffer observability issues when the UAV is undergoing degenerate motions, especially during hovering with yaw-rotation-only motion. In practice, absolute zero acceleration with yaw-only rotation is unlikely to happen, nevertheless it is a potential cause of failure that we need to consider.

The fisheye camera suffers image quality issues. The figure of the Modulation Transfer Function (MTF) of the fisheye lens we used is shown in Figure 31. The pixel quality near the image center is significantly better than those at the border of the image. This violates the illumination invariance assumption used in depth estimation, which further affects the quality of the depth image. In our future work, this lens property will be calibrated to improve the accuracy of the depth image.

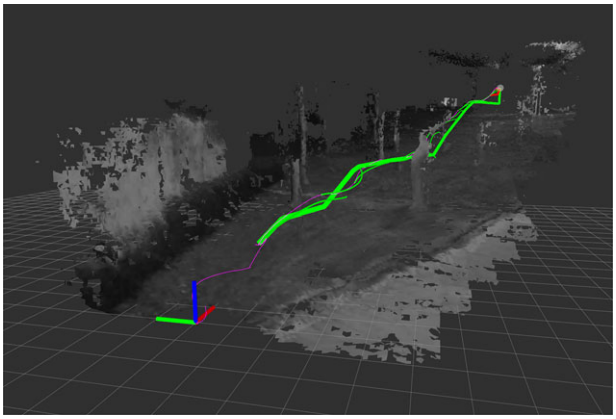
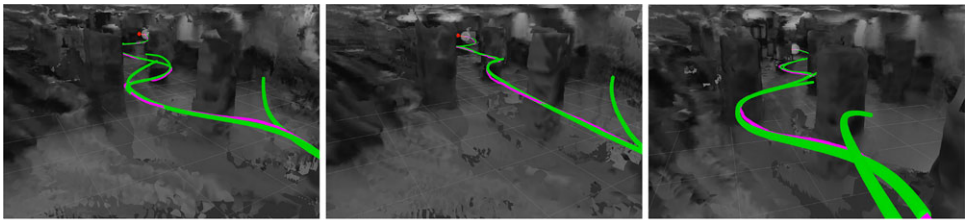


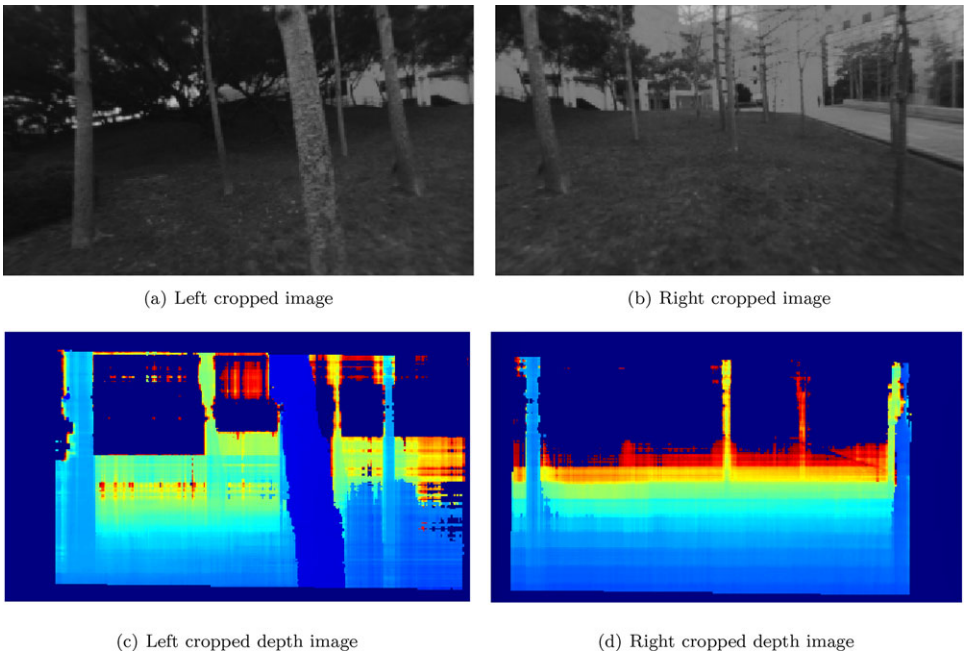
FIGURE 29 As the quadrotor flies forward, more depth images fuse into the TSDF and complete the global map. Under the 3D motion planning, our quadrotor flies through the forest avoiding all the trees that are detected by the large FOV monocular fisheye camera

Due to the property of our TSDF map, it is possible to construct hollow obstacles (i.e., obstacles that are not occupied in its interior). If our global path finder generates a guiding path right through these hollow obstacles due to insufficient observations, a potential failure may occur: the local planner can never find a feasible path to move out from the obstacle. However, as mentioned in Section 7.4, we have some



(a) (b) (c)

FIGURE 27 Three different demonstrations of the same dense indoor navigation experiment. Since the trajectories are optimized depending on the map, and the map is reconstructed depending on the quadrotor flying path, they are not totally the same but similar



(a) Left cropped image (b) Right cropped image

(c) Left cropped depth image (d) Right cropped depth image

FIGURE 30 The left and right ROI images with 320×184 resolution from the fisheye camera. (a) and (b) are ROI images. (c) and (d) are the corresponding depth images generated from the mapping module

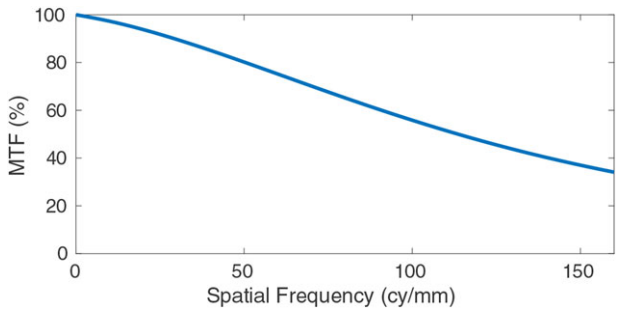


FIGURE 31 The MTF describes the image quality via distribution of the image optical system

strategies in case the local planner cannot find a feasible path to a planning target. Unfortunately, if the hollow obstacle is very large, the planning mission will fail.

9.2 | Conclusion

In this work, we present a real-time monocular visual-inertial dense mapping and autonomous navigation system. The whole system is implemented on a tight size and light weight quadrotor where all modules are processing onboard and in real time. By properly coordinating three major system modules—state estimation, dense mapping, and trajectory planning—we validate our system in both cluttered indoor and outdoor environments via multiple autonomous flight experiments. A tightly coupled monocular visual-inertial state estimator is develop for providing high-accuracy odometry, which is used for both feedback control and dense mapping. Our estimator supports on-the-fly initialization, and it is able online to estimate vehicle velocity, metric scale, and IMU biases. Without any prior knowledge of the environment, our dense mapping module utilizes a plane-sweeping-based method to extract a 3D cost volume through temporal aggregation on synchronized camera poses. After semiglobal optimization and postprocessing, a dense depth image is calculated and fed into our uncertainty-aware TSDF fusion approach, from which a live dense 3D map is produced. Using this map, our planning module first generates an initial collision-free trajectory based on our sampling-based path searching method. A gradient-based optimization method is then applied to ensure trajectory smoothness and dynamic feasibility. Following the trend of rapid increases in mobile computing power, we believe our minimum sensing sensor setup suggests a feasible solution to fully autonomous miniaturized aerial robots.

REFERENCES

1. Schmid K, Lutz P, Tomić T, et al. Autonomous vision-based micro air vehicle for indoor and outdoor navigation. *J Field Robot*. 2014;31(4):537–570.
2. Hornung A, Wurm KM, Bennewitz M, et al. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Auton. Robots*. 2013;34(3):189–206.
3. Fraundorfer F, Heng L, Honegger D, et al. Vision-based autonomous mapping and exploration using a quadrotor mav. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE; 2012:4557–4564.
4. Forster C, Pizzoli M, Scaramuzza D. Svo: Fast semi-direct monocular visual odometry. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE; 2014a:15–22.
5. Pizzoli M, Forster C, Scaramuzza D. Remode: Probabilistic, monocular dense reconstruction in real time. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE; 2014:2609–2616.
6. Faessler M, Fontana F, Forster C, et al. Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle. *J Field Robot*. 2016;33(4):431–450.

7. Loianno G, Mulgaonkar Y, Brunner C, et al. Smartphones power flying robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE; 2015:1256–1263.
8. Daftry S, Zeng S, Khan A, et al. Robust monocular flight in cluttered outdoor environments. arXiv:1604.04779; 2016.
9. Engel J, Schöps T, Cremers D. Lsd-slam: Large-scale direct monocular slam. In *Proceedings of the European Conference on Computer Vision*. Springer; 2014:834–849.
10. Fang Z, Yang S, Jain S, et al. Robust autonomous flight in constrained and visually degraded shipboard environments. *J Field Robot*. 2017;34(1):25–52.
11. Bry A, Bachrach A, Roy N. State estimation for aggressive flight in gps-denied environments using onboard sensing. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE; 2012:1–8.
12. Giusti A, Guzzi J, Cireşan DC, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robot Autom Lett*. 2016;1(2):661–667.
13. Michels J, Saxena A, Ng AY. High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proceedings of the International Conference on Machine Learning*. ACM; 2005:593–600.
14. Mei C, Rives P. Single view point omnidirectional camera calibration from planar grids. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE; 2007:3945–3950.
15. Heng L, Li B, Pollefeys M. Camodocal: Automatic intrinsic and extrinsic calibration of a rig with multiple generic cameras and odometry. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE; 2013:1793–1800.
16. Shen S, Michael N, Kumar V. Tightly-coupled monocular visual-inertial fusion for autonomous flight of rotorcraft MAVs. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE; 2015:5303–5310.
17. Yang Z, Shen S. Monocular visual–inertial state estimation with online initialization and camera–imu extrinsic calibration. *IEEE Trans Autom Sci Eng*. 2017;14(1):39–51.
18. Hesch JA, Kottas DG, Bowman SL, et al. Consistency analysis and improvement of vision-aided inertial navigation. *IEEE Trans Robot*. 2014;30(1):158–176.
19. Li M, Mourikis A. High-precision, consistent EKF-based visual-inertial odometry. *Int J Robot Res*. 2013;32(6):690–711.
20. Scaramuzza D, Achtelik M, Doitsidis L, et al. Vision-controlled micro flying robots: From system design to autonomous navigation and mapping in GPS-denied environments. *IEEE Robot Autom Mag*. 2014;21(3).
21. Leutenegger S, Lynen S, Bosse M, et al. Keyframe-based visual–inertial odometry using nonlinear optimization. *Int J Robot Res*. 2014;34(3):314–334.
22. Huang AS, Bachrach A, Henry P, et al. Visual odometry and mapping for autonomous flight using an RGB-D camera. In *Proceedings of the International Symposium of Robotics Research*. Flagstaff, AZ; 2011.
23. Mourikis AI, Roumeliotis SI. A multi-state constraint Kalman filter for vision-aided inertial navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE; 2007:3565–3572.
24. Kelly J, Sukhatme GS. Visual-inertial sensor fusion: Localization, mapping and sensor-to-sensor self-calibration. *Int J Robot Res*. 2011;30(1):56–79.
25. Jones ES, Soatto S. Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. *Int J Robot Res*. 2011;30(4):407–430.
26. Lynen S, Achtelik MW, Weiss S, et al. A robust and modular multi-sensor fusion approach applied to mav navigation. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE; 2013:3923–3929.
27. Indelman V, Williams S, Kaess M, et al. Information fusion in navigation systems via factor graph based incremental smoothing. *Robot Auton Syst*. 2013;61(8):721–738.
28. Shen S, Mulgaonkar Y, Michael N, et al. Initialization-free monocular visual-inertial estimation with application to autonomous MAVs. In *Proceedings of the International Symposium on Experimental Robotics*. Marrakesh, Morocco; 2014.
29. Martinelli A. Closed-form solution of visual-inertial structure from motion. *Int J Comput Vis*. 2014;106(2):138–152.
30. Kaiser J, Martinelli A, Fontana F, et al. Simultaneous state initialization and gyroscope bias calibration in visual inertial aided navigation. *IEEE Robot Autom Lett*. 2017;2(1):18–25.
31. Faessler M, Fontana F, Forster C, et al. Automatic re-initialization and failure recovery for aggressive flight with a monocular vision-based quadrotor. In *2015 IEEE International Conference on Robotics and Automation*. IEEE; 2015b:1722–1729.
32. Forster C, Pizzoli M, Scaramuzza D. SVO: Fast semi-direct monocular visual odometry. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE; 2014b.
33. Lupton T, Sukkarieh S. Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions. *IEEE Trans Robot*. 2012;28(1):61–76.
34. Forster C, Carlone L, Dellaert F, et al. IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *Proceedings of Robotics: Science and Systems*. Rome; 2015.
35. Lucas BD, Kanade T. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*. Vancouver, Canada; 1981:24–28.
36. Hartley R, Zisserman A. *Multiple View Geometry in Computer Vision*. Cambridge, UK: Cambridge University Press; 2003.
37. Nistér D. An efficient solution to the five-point relative pose problem. *IEEE Trans Pattern Anal Machine Intell*. 2004;26(6):756–770.
38. Triggs B, McLauchlan PF, Hartley RI, et al. Bundle adjustment—A modern synthesis. In *International Workshop on Vision Algorithms*. Springer; 1999:298–372.
39. Ling Y, Shen S. High-precision online markerless stereo extrinsic calibration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE; 2016:1771–1778.
40. Yang Z, Gao F, Shen S. Real-time monocular dense mapping on aerial robots using visual-inertial fusion. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 2017. Available at https://drive.google.com/open?id=0B0q9-c-ZdVPiLXBINU1FWm5lY1U.
41. Newcombe RA, Izadi S, Hilliges O, et al. Kinectfusion: Real-time dense surface mapping and tracking. In *Proceedings of the International Symposium on Mixed and Augmented Reality*. IEEE; 2011a:127–136.
42. Ok K, Greene WN, Roy N. Simultaneous tracking and rendering: Real-time monocular localization for mavs. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE; 2016:4522–4529.
43. Oleynikova H, Taylor Z, Fehr M, et al. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. arXiv:1611.03631; 2016b.
44. Hirschmuller H. Stereo processing by semiglobal matching and mutual information. *IEEE Trans Pattern Anal Mach Intell*. 2008;30(2):328–341.
45. Zbontar J, LeCun Y. Stereo matching by training a convolutional neural network to compare image patches. *J Mach Learn Res*. 2016;17(1-32):2.

46. Ha H, Im S, Park J, et al. High-quality depth from uncalibrated small motion clip. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*. IEEE; 2016:5413–5421.
47. Newcombe RA, Lovegrove SJ, Davison AJ. Dtam: Dense tracking and mapping in real-time. In *Proceedings of the IEEE International Conference on Computer Vision*. IEEE; 2011b:2320–2327.
48. Collins RT. A space-sweep approach to true multi-image matching. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*. IEEE; 1996:358–363.
49. Akbarzadeh A, Frahm J-M, Mordohai P, et al. Towards urban 3d reconstruction from video. In *Third International Symposium on 3D Data Processing, Visualization, and Transmission*. IEEE; 2006:1–8.
50. Klingensmith M, Dryanovski I, Srinivasa S, et al. Chisel: Real time large scale 3d reconstruction onboard a mobile device using spatially hashed signed distance fields. In *Proceedings of Robotics: Science and Systems*. 2015.
51. Lorensen WE, Cline HE. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM Siggraph Computer Graphics*. ACM; 1987;21:163–169.
52. LaValle S. Rapidly-exploring random trees: A new tool for path planning; 1998.
53. Karaman S, Frazzoli E. Sampling-based algorithms for optimal motion planning. *Int J Robot Res*. 2011;30:846–894.
54. Mellinger D, Kumar V. Minimum snap trajectory generation and control for quadrotors. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE; 2011:2520–2525.
55. Gao F, Shen S. Online quadrotor trajectory generation and autonomous navigation. In *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics*. IEEE; 2016.
56. Oleynikova H, Burri M, Taylor Z, et al. Continuous-time trajectory optimization for online uav replanning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE; 2016a:5332–5339.
57. Gammell JD, Srinivasa SS, Barfoot TD. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE; 2014:2997–3004.
58. Taeyoung L, Melvin L, Harris MN. Geometric tracking control of a quadrotor uav on SE(3). In *Proceedings of the IEEE Control and Decision Conference*. IEEE; 2010:5420–5425.
59. Ratliff N, Zucker M, Bagnell JA, et al. Chomp: Gradient optimization techniques for efficient motion planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE; 2009:489–494.
60. Richter C, Bry A, Roy N. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Proceedings of the International Symposium of Robotics Research*. Springer, Singapore; 2013:649–666.
61. Liu S, Watterson M, Mohta K, et al. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *IEEE Robot Autom Lett*.; 2017;2(3):1688–1695.
62. Handa A, Newcombe RA, Angeli A, et al. Real-time camera tracking: When is high frame-rate best? In *Proceedings of the European Conference on Computer Vision*. Springer; 2012:222–235.

SUPPORTING INFORMATION

Additional Supporting Information may be found online in the supporting information tab for this article.

How to cite this article: Lin Y, Gao F, Qin T, et al. Autonomous aerial navigation using monocular visual-inertial fusion. *J Field Robotics*. 2017;00:1–29. https://doi.org/10.1002/rob.21732

APPENDIX A

TABLE A1 Index to multimedia extensions

Extension	Media Type	Description
1	Video	It shows some of the experiments presented in this paper