

Autonomous Quadrotor Flight despite Rotor Failure with Onboard Vision Sensors: Frames vs. Events

Sihao Sun, Giovanni Cioffi, and Davide Scaramuzza

Abstract—Fault-tolerant control is crucial for safety-critical systems, such as quadrotors. State-of-the-art flight controllers can stabilize and control a quadrotor even when subjected to the complete loss of a rotor. However, these methods rely on external sensors, such as GPS or motion capture systems, for state estimation. In this paper, we propose the first algorithm that combines fault-tolerant control and onboard vision-based state estimation to achieve position control of a quadrotor subjected to complete failure of one rotor. Experimental validations show that our approach is able to accurately control the position of a quadrotor during a motor failure scenario, without the aid of any external sensors. The primary challenge to vision-based state estimation stems from the inevitable high-speed yaw rotation (over 20 rad/s) of the damaged quadrotor, causing motion blur to cameras, which is detrimental to visual inertial odometry (VIO). We compare two types of visual inputs to the vision-based state estimation algorithm: standard frames and events. Experimental results show the advantage of using an event camera especially in low light environments due to its inherent high dynamic range and high temporal resolution. We believe that our approach will render autonomous quadrotors safer in both GPS denied or degraded environments.

SOURCE CODE AND VIDEO

The source code of both our controller and VIO algorithm is available at: https://github.com/uzh-rpg/fault_tolerant_control

A video of the experiments is available at: <https://youtu.be/Ww8u0KH7Ugs>

I. INTRODUCTION

Quadrotors are safety-critical systems, which are vulnerable to motor failures. Fault-tolerant flight control of quadrotors is a feasible solution since only software adaptation is required, which is an obvious advantage over adding rotor redundancy or parachutes. Previous works have achieved autonomous flight of a quadrotor subjected to complete failure of a single rotor [1], [2], and even in high-speed flight conditions where aerodynamic disturbances are apparent [3], [4]. However, these methods rely on external sensors, such as GPS or motion capture systems, for position tracking, which completely eliminate or alleviate the effects of state estimation errors. To improve quadrotor safety in GPS denied or degraded environments, we need to resort to fully onboard solutions, such as vision-based state estimation.

Complete failure of a rotor results in a fast spinning motion (> 20 rad/s) of the quadrotor [5]. This high-speed motion brings a significant challenge to onboard state-estimation methods. First, in vision-based estimators, it causes motion blur (bottom left plot in Fig. 1). Such motion blur deteriorates feature detection and tracking and subsequently degrades visual-inertial odometry (VIO) performance, especially in

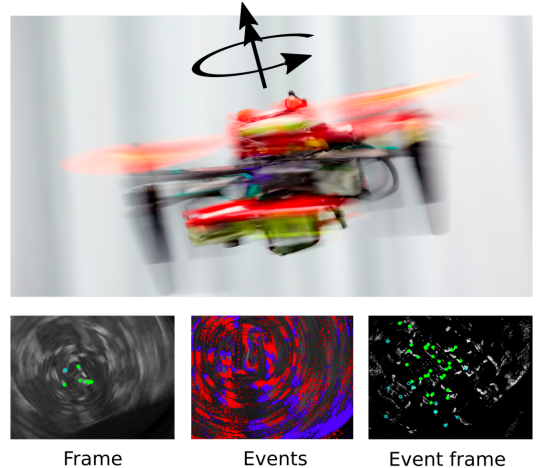


Fig. 1: Controlling a quadrotor with complete failure of a rotor causes fast yaw rotation, over 20 rad/s (top figure). Bottom figures show a standard frame and events captured by an onboard event camera. **Bottom Left**: standard frame with motion blur. **Bottom Center**: Events only (blue: positive events, red: negative events). **Bottom Right**: Event frame generated from events. Blue circles are detected features, green dots indicate tracked features.

low-light environments. Secondly, a large centrifugal acceleration read by the inertial measurement unit (IMU) often causes large errors in commonly-used attitude estimators. These problems need to be resolved to achieve autonomous quadrotor flight despite rotor failure, using only onboard sensors.

The contribution of this work is proposal of a state-estimation pipeline combining measurements from an IMU, a range sensor, and a vision sensor that can be either a standard camera or an event camera [6]. We validate the capability of this algorithm of providing reliable pose and velocity estimates of a quadrotor to realize fault-tolerant flight control. We show that an event camera becomes advantageous in low-light environments because the sensor does not suffer from motion blur and has a very high dynamic range.

II. ONBOARD STATE ESTIMATION

To achieve autonomous flight with rotor failure, the state estimator provides orientation and position information using only onboard sensors, including an IMU, a downward looking range sensor, and a downward looking camera. The camera can be either a standard camera or an event camera.

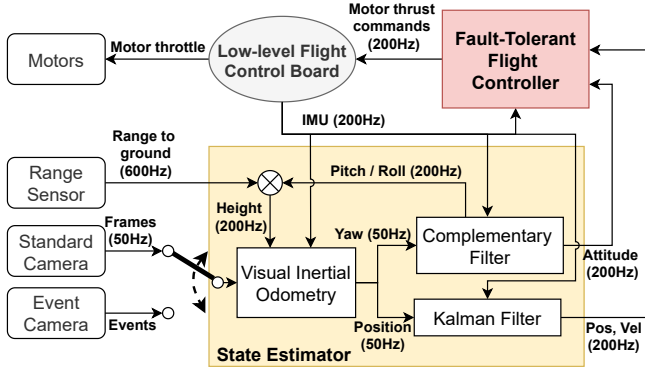


Fig. 2: General diagram of the onboard state estimator and the fault-tolerant controller.

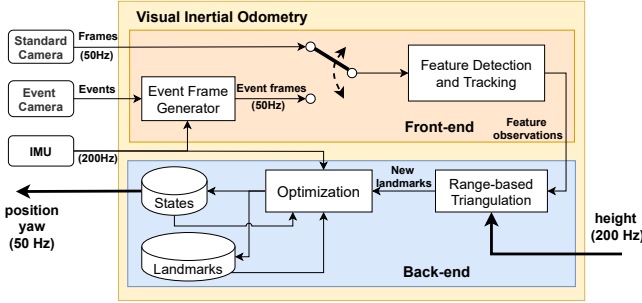


Fig. 3: Diagram of the visual-inertial odometry (either event based or frame based).

The block diagram of our system is given in Fig. 2. A range-sensor-aided monocular VIO algorithm provides pose estimates of the camera at 50 Hz. These estimates are fused with the IMU from a low-level flight control board located at the center of gravity, using a Kalman filter and a complementary filter, to provide position, velocity, and orientation estimates at 200 Hz for the fault-tolerant flight controller. A rotation compensated complementary filter is proposed for orientation estimation instead of directly using the orientation from the VIO. This can improve the robustness of the algorithm in case of loss of feature tracks.

A. Visual Inertial Odometry

The proposed VIO provides position and yaw estimations. It can use either standard frames or events as visual inputs. A block diagram of the algorithm is given in Fig. 3.

1) *Front-end*: If events are selected as visual input, we first of all generate synthetic event frames [7]. Each event frame $I_k(\mathbf{x})$ is aggregated from events from a spatio-temporal window W_k :

$$I_k(\mathbf{x}) = \sum_{e_j \in W_k} p_j \delta(\mathbf{x} - \mathbf{x}'_j), \quad (1)$$

where function $\delta(\cdot)$ is the Kronecker delta, p_j is the polarity of a certain event represented by e_j , \mathbf{x}'_j is the corrected event position considering the motion of the camera:

$$\mathbf{x}'_j = \pi \left(T_{t_k, t_j} \left(Z(\mathbf{x}_i) \pi^{-1}(\mathbf{x}_i) \right) \right). \quad (2)$$

where $\pi(\cdot)$ is the camera projection model, T_{t_k, t_j} is the transformation of camera pose between t_k and t_j , $Z(\mathbf{x}_i)$ is the scene depth approximated by the range sensor.

As the rotation of the damaged quadrotor in this time window is more dominant than the linear motion, we use a pure rotation transformation to approximate $T_{t_k, t_{k-1}}$, which is generated by integrating angular rate measurements from the gyroscope. Then T_{t_k, t_j} is approximated by a linear interpolation $T_{t_k, t_j} = T_{t_k, t_{k-1}}(t_j - t_{k-1}) / (t_k - t_{k-1})$.

Then we use a FAST feature detector [8] and Lucas-Kanade tracker [9] to detect and track features. If a feature has been tracked over 3 consecutive frames, it is determined as persistent and is triangulated. The corresponding landmark is added to the map. These settings are the same as applied in [10] and [7].

If standard frames are selected as visual input, we simply replace the synthetic event frames in the above procedure.

2) *Range-Based Landmark Triangulation*: Since the damaged quadrotor motion is mostly rotation, triangulating landmarks based on the disparity is inaccurate. For this reason, we use a downward range sensor to detect the range of the camera to the ground, where most features are detected. We also assume that all landmarks lie within the ground plane. For the j -th landmark whose position in the inertial coordinate frame is defined as \mathbf{p}_j we have

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \mathbf{R}_{\text{CI}} (\mathbf{p}_j - \mathbf{p}_c), \quad (3)$$

$$p_{c,z} - p_{j,z} = \hat{h} = r \cos \theta \cos \phi,$$

where u, v are observations of the landmark in the last (event) frame, \mathbf{p}_c is the position of the camera in the inertial coordinate frame, \mathbf{K} is the camera calibration matrix, \mathbf{R}_{CI} is the rotation matrix from camera frame to the inertial frame, \hat{h} is the height estimate, r is the range measurement from the range sensor.

As \mathbf{R}_{CI} and \mathbf{p}_c are obtained from the backend optimization, ϕ and θ are estimated from the complementary filter, we can subsequently solve \mathbf{p}_j from (3). The position estimates of triangulated landmarks are then used as initial guess for the nonlinear optimization problem in the backend.

3) *Back-end*: We use a keyframe-based fixed-lag smoother inspired by [11] to estimate the pose of the camera. The optimization cost function is formulated as

$$J(\mathbf{X}) = \sum_{k=1}^K \sum_{j \in J(k)} \| \mathbf{e}_v^{j,k} \|^2_{\mathbf{w}_v^{j,k}} + \sum_{k=1}^{K-1} \| \mathbf{e}_i^k \|^2_{\mathbf{w}_i^k} + \sum_{k=1}^K \sum_{j \in J(k)} w_h^{j,k} (p_{c,z} - \hat{h} - p_{j,z})^2, \quad (4)$$

where k is the frame index, K denotes number of frame in the sliding window, j is the landmark index, $J(k)$ is the set containing all the visible landmarks from the frame k . The first and second term in (4) represent reprojection error and inertial error respectively. The optimization variables are

the states of the K frames in the sliding window, which are represented by X and include position, velocity, orientation, and IMU biases.

Differently from the optimization-based back-end in [11], we add the third term in the cost function (4), where $w_h^{j,k}$ is the weight. It forces the vertical differences between the position of the camera and the observed landmark to be equal to the height estimate \hat{h} from the range sensor. By this means we add additional scale information from the range sensor while the IMU based scale information becomes less reliable in this fast rotational motion dominated task.

The optimization is run when a new (event) frame is generated. To improve computation efficiency, we do not perform marginalization and discard the states and measurements outside the sliding window. We use the Google Ceres Solver [12] to solve this nonlinear least square problem.

III. EXPERIMENTS

A. Hardware Descriptions

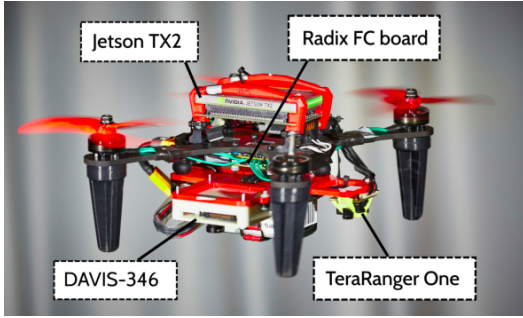
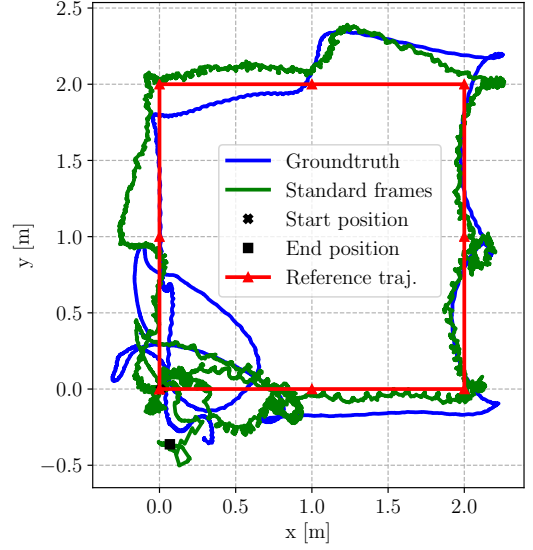


Fig. 4: Photo of a quadrotor flying with three rotors, where an event camera is used for state estimation.

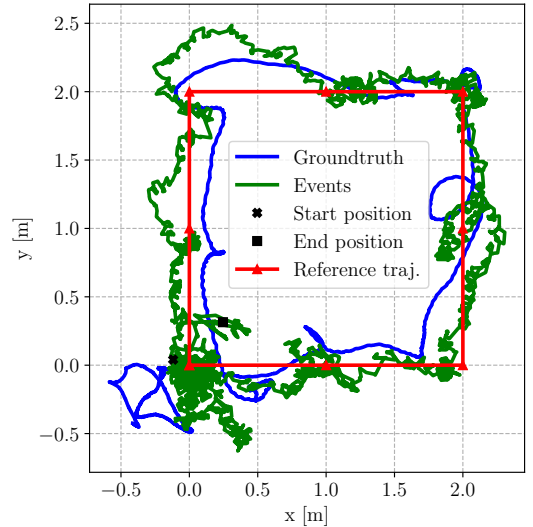
Fig. 4 shows the tested quadrotor. The state estimation and the control algorithm are run on a Nvidia Jetson TX2, which contains a quad-core 2.0 Hz CPU and a dual-core 2.5 Hz CPU running Ubuntu 18.04 and ROS [13]. The motor thrust commands from the control algorithm are sent to motors through a Radix FC flight control board, which runs a self-built firmware that also sends the IMU measurements to the TX2 at 200 Hz. We used TeraRanger One, a LED time-of-flight range sensor, to measure the distance to the ground. Both standard and event cameras are facing downward. They both use a 110° field-of-view lens. For the event camera, we use an Inivation DAVIS-346 with a resolution of 346×240 pixels. For the standard camera, we use a mvBlueFox-220w [14] with a resolution of 376×240 pixels, chosen intentionally to be close to that of the event camera to enable a fair comparison. It is worth noting that the maximum gain (12 dB) of the mvBlueFox-200w camera is used to minimize the required exposure time, which is found essential in reducing the motion blur in the standard frames.

B. Closed-loop Flight Validation

We conducted setpoint tracking tasks in the closed-loop flight experiment to validate the entire algorithm, including



(a) Closed-loop flight trajectory using **standard frames**.








(b) Closed-loop flight trajectory using **events**.

Fig. 5: Top view of the closed-loop flight trajectories. Ground-truth from the motion capture system (blue), estimated trajectory (green), and the reference trajectory (red) including 9 setpoints.

the vision-based state estimator and the fault-tolerant flight controller. During the test, the quadrotor took off with four rotors. The VIO was initialized while the drone was in hovering. Then, we switched off one rotor and the fault-tolerant flight controller started controlling the quadrotor. As shown in Fig. 5a, nine setpoints formed into a square were given to the flight controller in steps of 5 seconds. The damaged quadrotor then flew a square trajectory by tracking these setpoints.

Two different tests were performed where standard frames and events were respectively used as visual input to the state estimator. Fig. 5a shows the closed-loop flight result using standard frames, where the estimated trajectory from the VIO

TABLE I: Position tracking accuracy (RMSE) with state estimators using standard frames or events in different light conditions. For standard frames, gains of the camera are set as 12 dB. Env lux: environment illuminance. Cam lux: illuminance at the camera lens. The left column shows photos of the test environment.

	Environment Illuminance		Standard Frames		Events	
	Env lux	Cam lux	Exposure (ms)	RMSE (m)	Num of events	RMSE (m)
	500	71.5	2	0.50	15000	0.48
	100	18.0	8	0.93	15000	0.42
	50	7.3	12	∞	6000	0.58
	10	2.3	12	∞	6000	0.89
	1	0.2	12	∞	6000	∞

in the inertial frame, the ground truth trajectory measured by the motion capture system, and the reference trajectory are presented. The difference between the real trajectory and the reference is caused by the position estimation error and the controller tracking error. Although the tracking performance is not perfect, it is sufficient for controlling a damaged quadrotor to a safe area for landing. Similarly, Fig. 5b shows another test where events are used as visual input to the state estimator. Both tests were conducted in a well-illuminated environment (500 lux), which is a bright indoor office lighting condition [15].

C. Comparison between Frames and Events

In this section, we test the algorithm in different environment's lighting conditions, and compare between using frames and events as visual input. In these tests, we let the damaged quadrotor track a single setpoint (i.e., hovering). Then, we measure the root mean square error (RMSE) of the closed-loop position tracking to evaluate the performance of the entire algorithm. The exposure times of the standard camera are changed according to the environment brightness to capture frames with sufficient intensity for detecting and tracking features. For the event camera, we observe that the number of events generated in a fixed time window is smaller in a darker environment. Hence, we accordingly reduce the number of events needed to construct an event frame in low-light conditions.

Table I reports the position tracking RMSE when using standard frames or events in different lighting conditions. As can be observed, when the environment illuminance is around 500 lux, both frames and events can accomplish the task with similar tracking error. However, with standard frames as visual input, the tracking error doubles as the illuminance drops to 100 lux, and the damaged quadrotor crashes immediately when the illuminance gets lower than

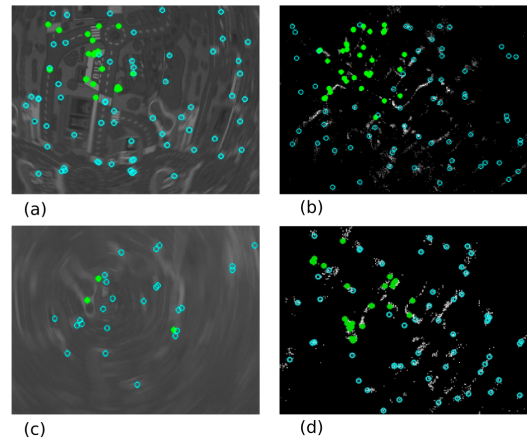


Fig. 6: Standard and event frames, including features (light blue circle are detected features, green dots are persistent features), in a bright (500 lux) and a dark (50 lux) environment. (a) Standard frame in the bright environment with 2 ms exposure time. (b) Event frame in the bright environment. (c) Standard frame in the dark environment with 12 ms exposure time. (d) Event frame in the dark environment.

100 lux. By contrast, with events as visual input, the damaged quadrotor can even fly when the illuminance is decreased to 10 lux. These comparisons clearly show the advantage of using an event camera for state estimation in low-light conditions.

Fig. 6 presents the standard frames and the event frames in a bright (500 lux) and a dark (50 lux) indoor environment, respectively. When the environment illuminance is 50 lux, a relatively long exposure time (12 ms) of the standard camera is required. Hence, the standard frames experience significant motion blur (Fig. 6c) owing to the quadrotor fast rotational motion caused by the motor failure. Although more than 20 features are detected from this blurry image, few of them are successfully tracked (i.e., persistent features) and added to the map, causing failure of the standard frame-based VIO. By contrast, the event frames are sharp enough for feature detection and tracking in both bright and dark conditions.

IV. CONCLUSIONS

In this work, we achieved the first autonomous flight of a quadrotor despite loss of a single rotor, using only onboard sensors. A new state estimation pipeline was proposed. Comparisons were made between different visual inputs to the proposed algorithm: standard frames and events. In a well-illuminated environment, we demonstrated that the algorithms using both forms of visual input can provide sufficiently accurate state estimates. However, in a relatively low-light environment with illuminance lower than 100 lux, the standard frames were affected by significant motion blur due to the fast rotation and the long exposure time required. By contrast, the event-based algorithm could stand closed-loop tests in a much darker environment (10 lux).

REFERENCES

- [1] M. W. Mueller and R. D’Andrea, “Relaxed hover solutions for multicopters: Application to algorithmic redundancy and novel vehicles,” *Int. J. Robot. Research*, vol. 35, no. 8, pp. 873–889, 2016.
- [2] J. Stephan, L. Schmitt, and W. Fichter, “Linear parameter-varying control for quadrotors in case of complete actuator loss,” *Journal of Guidance, Control, and Dynamics*, vol. 41, no. 10, pp. 2232–2246, 2018.
- [3] S. Sun, L. Sijbers, X. Wang, and C. de Visser, “High-Speed Flight of Quadrotor Despite Loss of Single Rotor,” *IEEE Robot. Autom. Lett. (RA-L)*, vol. 3, no. 4, pp. 3201–3207, 2018.
- [4] S. Sun, X. Wang, Q. Chu, and C. d. Visser, “Incremental nonlinear fault-tolerant control of a quadrotor with complete loss of two opposing rotors,” *IEEE Trans. Robot.*, pp. 1–15, 2020.
- [5] A. Freddi, A. Lanzon, and S. Longhi, “A feedback linearization approach to fault tolerance in quadrotor vehicles,” *IFAC proceedings volumes*, vol. 44, no. 1, pp. 5413–5418, 2011.
- [6] G. Gallego, T. Delbruck, G. M. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, “Event-based vision: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020.
- [7] H. Rebecq, T. Horstschaefer, and D. Scaramuzza, “Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization,” *British Machine Vision Conf. (BMVC)*, 2017.
- [8] E. Rosten and T. Drummond, “Machine learning for high-speed corner detection,” in *European Conf. on Comp. Vis. (ECCV)*, pp. 430–443, Springer, 2006.
- [9] B. D. Lucas, T. Kanade, *et al.*, “An iterative image registration technique with an application to stereo vision,” 1981.
- [10] A. R. Vidal, H. Rebecq, T. Horstschäfer, and D. Scaramuzza, “Ultimate slam? combining events, images, and imu for robust visual slam in hdr and high-speed scenarios,” *IEEE Robot. Autom. Lett. (RA-L)*, vol. 3, no. 2, pp. 994–1001, 2018.
- [11] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, “Keyframe-based visual-inertial odometry using nonlinear optimization,” *Int. J. Robot. Research*, vol. 34, no. 3, pp. 314–334, 2015.
- [12] N. S. S. Agarwal, “Ceres solver.” <http://ceres-solver.org/>. Accessed: 2020-10-14.
- [13] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.
- [14] “Matrix vision mvbluefox camera.” <https://www.matrix-vision.com/USB2.0-industrial-camera-mvbluefox.html>. Accessed: 2020-12-19.
- [15] “Lux-wikipedia.” <https://en.wikipedia.org/wiki/Lux>. Accessed: 2020-10-14.