Visual model-predictive localization for computationally efficient autonomous racing of a 72-g drone

Li, Shuo; van der Horst, Erik; Duernay, Philipp; De Wagter, Christophe; de Croon, Guido C.H.E.

DOI
10.1002/rob.21956

Publication date
2020

Document Version
Final published version

Published in
Journal of Field Robotics

Received: 15 May 2019 | Revised: 25 February 2020 | Accepted: 6 April 2020

DOI: 10.1002/rob.21956

REGULAR ARTICLE

WILEY

Visual model-predictive localization for computationally efficient autonomous racing of a 72-g drone

Shuo Li | **Erik van der Horst** | **Philipp Duernay** | **Christophe De Wagter** | **Guido C. H. E. de Croon**

Micro Air Vehicle Lab, Delft University of Technology, Delft, The Netherlands

Correspondence
Shuo Li, Micro Air Vehicle Lab, Delft University of Technology, 2629HS Delft, The Netherlands.
Email: s.li-4@tudelft.nl

Abstract

Drone racing is becoming a popular e-sport all over the world, and beating the best human drone race pilots has quickly become a new major challenge for artificial intelligence and robotics. In this paper, we propose a novel sensor fusion method called visual model-predictive localization (VML). Within a small time window, VML approximates the error between the model prediction position and the visual measurements as a linear function. Once the parameters of the function are estimated by the RANSAC algorithm, this error model can be used to compensate the prediction in the future. In this way, outliers can be handled efficiently and the vision delay can also be compensated efficiently. Theoretical analysis and simulation results show the clear advantage compared with Kalman filtering when dealing with the occasional large outliers and vision delays that occur in fast drone racing. Flight tests are performed on a tiny racing quadrotor named "Trashcan," which was equipped with a Jevois smart camera for a total of 72 g. An average speed of 2 m/s is achieved while the maximum speed is 2.6 m/s. To the best of our knowledge, this flying platform is currently the smallest autonomous racing drone in the world, while still being one of the fastest autonomous racing drones.

KEYWORDS

autonomous drone race, visual model-predictive localization

1 | INTRODUCTION

Drones, especially quadrotors, are transformed by enthusiasts in spectacular racing platforms. After years of development, drone racing has become a major e-sports, where the racers fly their drones in a preset course at high speed. It was reported that an experienced first-person view (FPV) racer can achieve speeds up to 190 km/h when sufficient space is available. The quadrotor itself uses an inertial measurement unit (IMU) to determine its attitude and rotation rates, allowing it to execute the human's steering commands. The human mostly looks at the images and provides the appropriate steering commands to fly through the track as fast as possible. The advance in research areas such as computer vision, artificial intelligence, and control raises the question: would drones not be able to fly faster than human pilots if they flew completely by themselves? Until now, this is an open question. In 2016, the world's first autonomous drone race was held at IROS 2016 (Moon, Sun, Baltes, & Kim, 2017), which became an annual event trying to answer this question (Figure 1).

We focus on developing computationally efficient algorithms and extremely light-weight autonomous racing drones that have the same or even better performance than currently existing larger drones. We believe that these drones may be able to fly faster, as the gates will

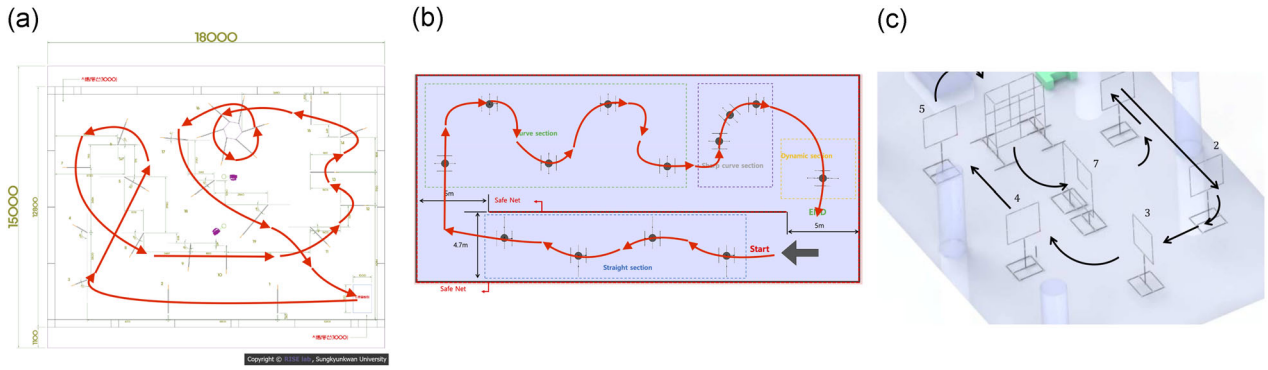

FIGURE 1 The IROS autonomous drone race track over the years 2016–2018 (a–c). The rules have always been the same. Flight is to be fully autonomous, so there can be no human intervention. The drone that passes through most subsequent gates in the track wins the race. When the number of passed gates is the same, or the track is fully completed, the fastest drone wins the race (a) IROS 2016 drone race track; (b) IROS 2017 drone race track; (c) IROS 2018 drone race track [Color figure can be viewed at wileyonlinelibrary.com]

be relatively larger for them. Moreover, a cheap, light-weight solution to drone racing would allow many people to use autonomous drones for training their racing skills. When the autonomous racing drone becomes small enough, people may even practice with such drones in their own home.

Autonomous drone racing is indebted to earlier work on agile flight. Initially, quadrotors made agile maneuvers with the help of external motion capture systems (Mellinger & Kumar, 2011; Mellinger, Michael, & Kumar, 2012). The most impressive feats involved passing at high speeds through gaps and circles. More recently, various researchers have focused on bringing the necessary state estimation for these maneuvers onboard. Loianno, Brunner, McGrath, and Kumar (2017) plan an optimal trajectory through a narrow gap with difficult angles while using visual-inertial odometry (VIO) for navigation. The average maximum speed of their drone can achieve 4.5 m/s. However, the position of the gap is known accurately a priori, so no gap detection module is included in their research. Falanga, Mueggler, Faessler, and Scaramuzza (2017) have their research on flying a drone through a gap aggressively by detecting the gap with fully onboard resources. They fuse the pose estimation from the detected gap and onboard sensors to estimate the state. In their experiment, the platform with a forward-facing fish-eye camera can fly through the gap with 3 m/s. Sanket, Singh, Ganguly, Fermüller, and Aloimonos (2018) develop a solution for a drone to fly through arbitrarily shaped gaps without building an explicit three-dimensional model of a scene, using only a monocular camera.

Drone racing represents a larger, even more challenging problem than performing short agile flight maneuvers. The reasons for this are that (a) all sensing and computing has to happen on board, (b) passing one gate is not enough. Drone races can contain complex trajectories through many gates, requiring good estimation and (optimal) control also on the longer term, and (c) depending on the race, gate positions can change, other obstacles than gates can be present, and the environment is much less controlled than an indoor motion tracking arena.

One category of strategies for autonomous drone racing is to have an accurate map of the track, where the gates have to be in the same place. One of the participants of the IROS 2017 autonomous drone race, the Robotics and Perception Group, reached gate 8 in 35 s. In their approach, waypoints were set using the pre-defined map and VIO was used for navigation. A depth sensor was used for aligning the track reference system with the odometry reference system. NASA's JPL lab report in their research results that their drone can finish their race track in a similar amount of time as a professional pilot. In their research, a visual-inertial localization and mapping system is used for navigation and an aggressive trajectory connecting waypoints is generated to finish the track (Morrell et al., 2018). Gao et al. (2019) come up with a teach-and-repeat solution for drone racing. In the teaching phase, the surrounding environment is reconstructed and a flight corridor is found. Then, the trajectory can be optimized within the corridor and be tracked during the repeating phase. In their research, VIO is employed for pose estimation and the speed can reach 3 m/s. However, this approach is sensitive to changing environments. When the position of the gate is changed, the drone has to learn the environment again.

The other category of strategies for autonomous drone race employs coarser maps and is more oriented on gate detection. This category is more robust to displacements of gates. The winner of IROS 2016 autonomous drone race, Unmanned Systems Research Group, uses a stereo camera for detecting the gates (Jung, Cho, Lee, Lee, & Shim, 2018). When the gate is detected, a waypoint will be placed in the center of the gate and a velocity command is generated to steer the drone to be aligned with the gate. The winner of the IROS 2017 autonomous drone race, the INAOE team, uses metric monocular SLAM for navigation. In their approach, the relative waypoints are set and the detection of the gates is used to correct the drift of the drone (Moon et al., 2019). S. Li, Ozo, De Wagter, and de Croon (2018) combine gate detection with onboard IMU readings and a simplified drag model for navigation. With their approach, a Parrot Bebop 1 (420 g) can use its native onboard camera and processor to fly through 15 gates with 1.5 m/s along a narrow track in a basement full of exhibits. Kaufmann, Loquercio, et al. (2018) use a trained Convolutional Neural Network (CNN) to map the input images to the desired waypoint and the desired speed to approach it. With the generated waypoint, a trajectory through the gate can be determined and executed while VIO is

used for navigation. The winner of the IROS 2018 autonomous drone race, the Robotics and Perception Group, finished the track with 2 m/s (Kaufmann, Gehrig, et al., 2018). During the flight, the relative position of the gates and a corresponding uncertainty measure are predicted by a CNN. With the estimated position of the gate, the waypoints are generated, and a model-predictive controller (MPC) is used to control the drone to fly through the waypoints while VIO is used for navigation.

From the research mentioned above, it can be seen that many of the strategies for autonomous drone racing are based on generic, but computationally relatively expensive navigation methods such as VIO or SLAM. These methods require heavier and more expensive processors and sensors, which leads to heavier and more expensive drone platforms. Forgoing these methods could lead to a considerable gain in computational effort, but raises the challenge of still obtaining fast and robust flight.

In this paper, we present a solution to this challenge. In particular, we propose a visual model-predictive localization (VML) approach to autonomous drone racing. The approach does not use generic vision methods such as VIO and SLAM and is still robust to gate changes, while reaching speeds competitive to the currently fastest autonomous racing drones. The main idea is to rely as much as possible on a predictive model of the drone dynamics, while correcting the model and localizing the drone visually based on the detected gates and their supposed positions in the global map. To demonstrate the efficiency of our approach, we implement the proposed algorithms on a cheap, commercially available smart camera called "Jevois" and mount it on the "Trashcan" racing drone. The modified Trashcan weighs only 72 g and is able to fly the race track with high speed (up to 2.6 m/s)[1]. The vision-based navigation and high-level controller run on the Jevois camera while the low-level controller provided by the open source Paparazzi autopilot (Gati, 2013; Hattenberger, Bronz, & Gorraz, 2014) runs on the Trashcan. To the best of our knowledge, the presented drone is the smallest and one of the fastest autonomous racing drone in the world. Figure 2 shows the weight and the speed of our drone in comparison to the drones of the winners of the IROS autonomous drone races.

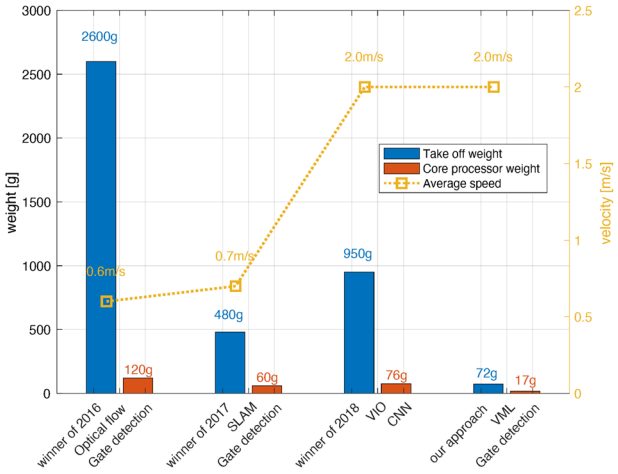

FIGURE 2 The weight and the speed of the approach proposed in this article and the winners' of IROS autonomous drone race. All weights are either directly from the articles or estimated from online specs of the used processors [Color figure can be viewed at wileyonlinelibrary.com]

2 | PROBLEM FORMULATION AND SYSTEM DESCRIPTION

2.1 | Problem formulation

In this study, we will develop a hardware and a software system that the flying platform can fly through a drone race track fully autonomously with high speed using only onboard resources. The racing track setup can be changed and the system should be adaptive to this change autonomously.

For visual navigation, instead of using SLAM or VIO, we directly use a computationally efficient vision algorithm for the detection of the racing gate to provide the position information. However, implementing such a vision algorithm on low-grade vision and processing hardware results in low frequency, noisy detections with occasional outliers. Thus, a filter should be employed to still provide high frequency and accurate state estimation. In Section 3, we first briefly introduce the "Snake Gate Detection" method and a pose estimation method used to provide position measurements. Then, we propose and analyze the novel VML technique that estimates the drone's states within a time window. It fuses the low-frequency onboard gate detections and high-frequency onboard sensor readings to estimate the position and the velocity of the drone. The control strategy to steer the drone through the racing track is discussed. The simulation result in Section 4 shows the comparison between the proposed filter and the Kalman filter in different scenarios with outliers and delay. In Section 5, we will introduce the flying experiment of the drone flying through a racing track with gate displacement, different altitude and moving gate during the flight. In Section 6, the generalization and the limitation of the proposed method are discussed. Section 7 concludes the article.

2.2 | System overview

To illustrate the efficiency of our approach, we use a small racing drone called Trashcan (Figure 3). This racing drone is designed for FPV racing with the Betaflight flight controller software. In our case, to fly this Trashcan autonomously, we replaced Betaflight by the Paparazzi open source autopilot for its flexibility of adding custom code, stable communication with the ground for testing code and active maintenance from the research community. In this article, the Paparazzi software only aims to provide a low-level controller. The main loop frequency is 2 kHz. We employ a basic complementary filter for attitude estimation and the attitude control loop is a

[1]The video of the experiment is available at https://www.youtube.com/playlist?list=PL_KSX9GOn2P8H0QvUZtLZSYggzQ2DFHCi

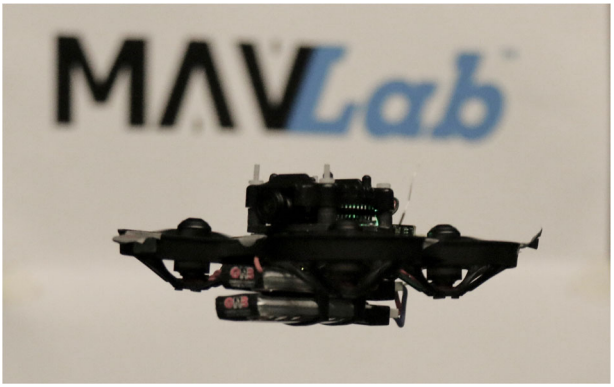

FIGURE 3 The flying platform. The Jevois is mounted on the Trashcan. The Trashcan provides power to the Jevois and they communicate with each other by the MAVLink protocol. The weight of the whole platform is only 72 g [Color figure can be viewed at wileyonlinelibrary.com]

cascade control including a rate loop and an attitude loop. For each loop, a P-controller is used. The details of Trashcan's hardware can be found in Table 1

For the high-level vision, flight planning and control tasks, we use a light-weight smart camera (17 g) called Jevois, which is equipped with a quad core ARM Cortex A7 processor and a dual core Mali-400 GPU. In our experiment, there are two threads running on the Jevois, one of which is for vision detection and the other one is for filtering and control (Figure 4a). In our case, the frequency of detecting gates ranges from 10 to 30 Hz and the frequency of filtering and control is set to 512 Hz. The Gate detection thread processes the images in sequence. When it detects the gate it will send a signal telling the other thread a gate is detected. The control and filtering thread keeps predicting the states and calculating control command in high frequency. It uses a novel filtering method, explained in Section 3, for estimating the state based on the IMU and the gate detections.

The communication between the Jevois and Trashcan is based on the MAVLink protocol with a baud rate of 115,200. Trashcan sends the Attitude and Heading Reference System (AHRS) estimation with a frequency of 512 Hz. And the Jevois sends the attitude and altitude commands to Trashcan with a frequency of 200 Hz. The software architecture of the flying platform can be found in Figure 4b.

In Figure 4b, the Gate detection and Pose estimation module first detects the gate and estimates the relative position between the drone and the gate. Next, the relative position will be sent to the Gate assignment module to be transferred to global position. With the global position measurements and the onboard AHRS reading, the proposed VML filter fuses them together to have accurate position and velocity estimation. Then, the Flight plan and high-level controller will calculate the desired attitude commands to steer the drone through the whole track. These attitude commands will be sent to the drone via MAVLink protocol. On the Trashcan drone, Paparazzi provides the low-level controller to stabilize the drone.

TABLE 1 The specifications of Trashcan's hardware

Weight	48 g (with the original camera)
Size	98 mm × 98 mm × 36 mm
Motor	TC0803 KV15000
MCU	STM32F4 (100 MHZ)
Receiver	FrSky D16

3 | ROBUST VML AND CONTROL

State estimation is an essential part of drones' autonomous navigation. For outdoor flight, fusing a GPS signal with onboard inertial sensors is a common way to estimate the pose of the drone (Santana, Brandao, & Sarcinelli-Filho, 2015). However, for indoor flight, a GPS signal is no longer available. Thus, off-board cameras (Lupashin et al., 2014), Ultra Wide Band Range beacons (Mueller, Hamer, & D'Andrea, 2015) or onboard cameras (McGuire, De Croon, De Wagter, Tuyls, & Kappen, 2017) can be used to provide the position or velocity measurements for the drone. The accuracy and time-delay of these types of infrastructure setups differ from each other. Hence, the different sensing setups have an effect on what type of filtering is best for each situation. The most commonly used state estimation technique in robotics is the Kalman filter and its variants, such as the Extended Kalman filter (EKF; Gross, Gu, Rhudy, Gururajan, & Napolitano, 2012; Santamaria-Navarro, Loianno, Solà, Kumar, & Andrade-Cetto, 2018; Weiss, Achtelik, Chli, & Siegwart, 2012). However, the racing scenario has properties that make it challenging for a Kalman filter. Position measurements from gate detections often are subject to outliers, have non-Gaussian noise, and can arrive at a low frequency. This makes the typical Kalman filter approach unsuitable because it is sensitive to outliers, is optimal only for Gaussian noise, and can converge slowly when few measurements arrive. In this section, we will propose a VML technique which is robust to low-frequency measurements with significant numbers of outliers. Subsequently, we will also present the control strategy for the autonomous drone race.

3.1 | Gate assignment

In this article, we use the "snake gate detection" and pose estimation technique as in S. Li et al. (2018). The basic idea of snake gate detection is searching for continuing pixels with the target color to find the four corners of the gate. Subsequently, a perspective *n*-point (PnP) problem is solved, using the position of the four corners in the image plane, the camera's intrinsic parameters, and the attitude estimation to solve the relative position between the drone and the *i*th gate at time *k*, $\Delta\bar{\mathbf{x}}_k^i = [\Delta\bar{x}_k^i, \Delta\bar{y}_k^i]$. Figure 5 shows this procedure, which is explained more in detail in S. Li et al. (2018). In most cases, when the light is even and the camera's auto exposure works properly, the gate in the image is continuous and the Snake gate detection

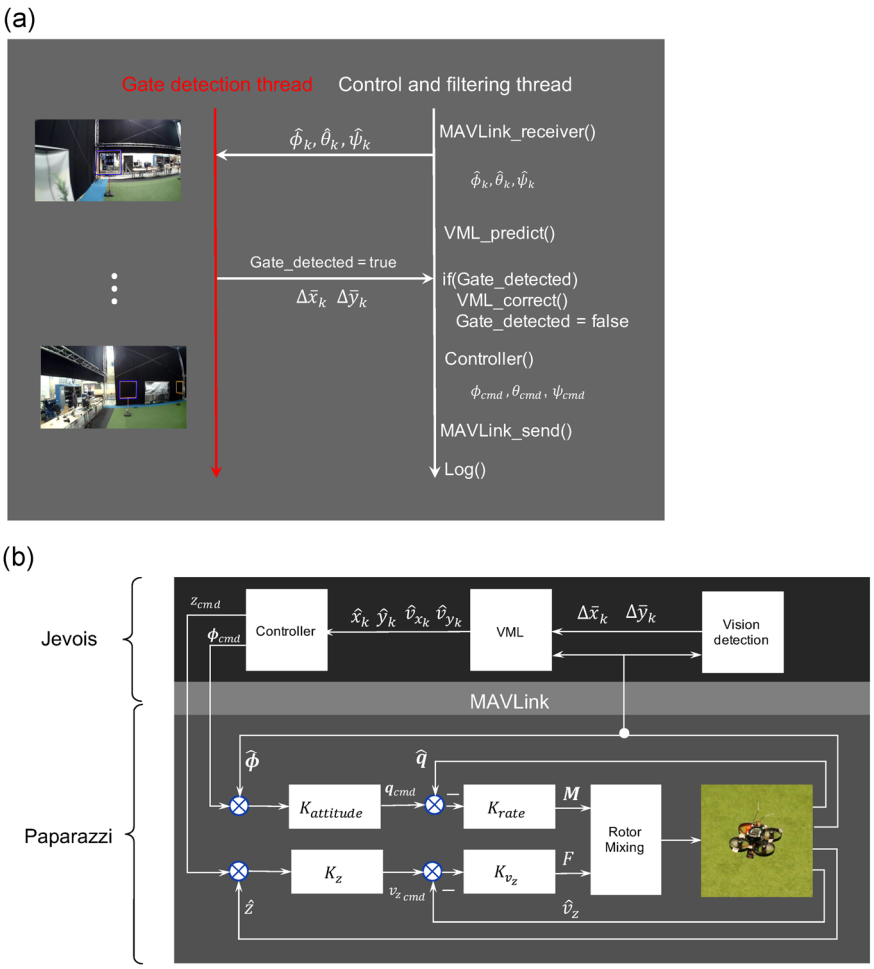

FIGURE 4 The architectures of the software on Jevois and the software of the whole flying platform. (a) The two threads structure running on Jevois. For the gate detection thread, the frequency of gate detection ranges from 10 to 30 Hz while the frequency of control and filtering thread is 512 Hz. (b) The software architecture of the UAV platform. The vision detection, filtering, and control are all running on Jevois. Paparazzi provides the low-level controller to stabilize the drone. VML, visual model-predictive localization [Color figure can be viewed at wileyonlinelibrary.com]

algorithm can detect the gate correctly. However, after an aggressive turn, such as a turn to a window, the camera cannot adapt to the new light condition immediately. In this case, Snake gate detection usually cannot detect the gate. Another failure case is that due to the uneven light condition or the similar color in the background, Snake gate detection may get interfered with. These situations make the searching stop in the middle of the bar or stop at the background pixels. Although we have some mechanism to prevent these false

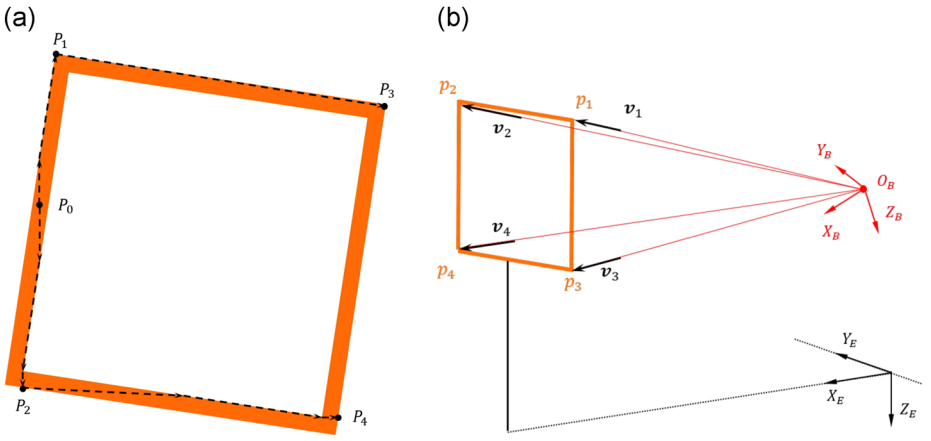

FIGURE 5 The Snake gate detection method and pose estimation method (S. Li et al., 2018). (a) Snake gate detection. From one point on the gate P_0, the Snake gate detection method first searches up and down, then left and right to find all the four corners of the gate. (b) When the four points of the gate are found, the relative position between the drone and the gate is calculated with the points' position, the camera's intrinsic parameters and the current attitude estimation [Color figure can be viewed at wileyonlinelibrary.com]

positive detections, there is still a small chance that a false positive happens. The negative effect is that outliers may appear which leads to a challenge for the filter and the controller.

Since for any race a coarse map of the gates is given a priori (cf. Figure 1), the position and the heading of gate i, $\mathbf{x}_g^i = \left[x_g^i, y_g^i, \psi_g^i\right]$ can be known roughly (Figure 6). We use the gates' positions to transfer the relative position $\Delta\bar{\mathbf{x}}_k^i$ measured by camera to a global position $\bar{\mathbf{x}}_k = [\bar{x}_k, \bar{y}_k]$ by Equation (1). In Equation (1), x_g^i, y_g^i and ψ_g^i are the position of the gate i which are known from the map.

$$\begin{bmatrix} \bar{x}_k \\ \bar{y}_k \end{bmatrix} = \begin{bmatrix} x_g^i \\ y_g^i \end{bmatrix} + \begin{bmatrix} \cos\psi_g^i & -\sin\psi_g^i \\ \sin\psi_g^i & \cos\psi_g^i \end{bmatrix} \begin{bmatrix} \Delta\bar{x}_k^i \\ \Delta\bar{y}_k^i \end{bmatrix}. \tag{1}$$

Here, we assume that the position of the gate is fixed. Any error experienced in the observations is then assumed to be due to estimation drift on the part of the drone. Namely, without generic VIO, it is difficult to make the difference between drone drift and gate displacements. If the displacements of the gates are moderate, this approach will work: after passing a displaced gate, the drone will see the next gate, and correct its position again. We only need a very rough map with the supposed global positions of the gates (Figure 6). Gate displacements only become problematic if after passing gate i the gate $i + 1$ would not be visible when following the path from the expected positions of gate i to gate $i + 1$.

At the IROS drone race, gates are identical, so for our position to be estimated well, we need to assign a detection to the right gate. For this, we rely on our current estimated global position $\hat{\mathbf{x}}_k = [\hat{x}_k, \hat{y}_k]$. When a gate is detected, we go through all the gates on the map

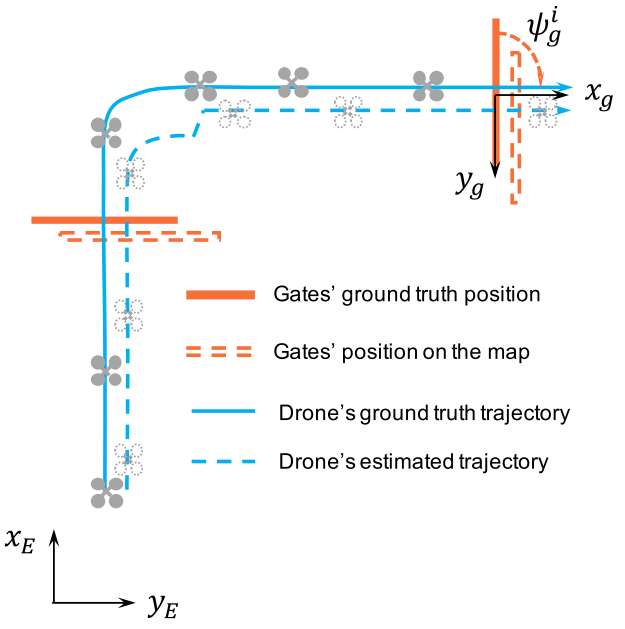

FIGURE 6 The gates are displaced. The drone uses the gate's position on the map to navigate. After passing through the first gate, it will use the second gate's position on the map for navigation. After seeing the second gate, the position of the drone will be corrected [Color figure can be viewed at wileyonlinelibrary.com]

using Equation (1) to calculate the predicted position $\bar{\mathbf{x}}_k^i = [\bar{x}_k^i, \bar{y}_k^i]$. Then, we calculate the distance between the predicted drone's position $\bar{\mathbf{x}}_k^i$ and its estimated position $\hat{\mathbf{x}}_k$ at time t_k by

$$\Delta d_k^i = \|\bar{\mathbf{x}}_k^i - \hat{\mathbf{x}}_k\|_2. \tag{2}$$

After going through all the gates, the gate with the predicted position closest to the estimated drone position is considered as the detected gate. At time t_k, the measurement position is determined by

$$\begin{aligned} j &= \text{argmin}_i \Delta d_k^i, \\ \bar{\mathbf{x}}_k &= \bar{\mathbf{x}}_k^j. \end{aligned} \tag{3}$$

The gate assignment technique (Figure 7) can help us obtain as much information on the drone's position as possible when a gate is detected. Namely, it can also use detections of other gates than the next gate, and allows to use multiple gate detections at the same time to improve the estimation. Still, this procedure will always output a global coordinate for any detection. Hence, false positive or inaccurate detections can occur and have to be dealt with by the state estimation filter.

3.2 | VML

The racing drone envisaged in this article has a forward-looking camera and an IMU. As explained in the previous section, the camera is used for localization in the environment, with the help of gate detections. Using a typical, cheap CMOS camera will result in relatively slow position updates from the gate detection, with occasional outliers. The IMU can provide high-frequency, and quite accurate attitude estimation by means of an AHRS. The accelerations can also be used in predicting the change in translational velocities of the drone. In traditional *inertial* approaches, the accelerations would be integrated. However, for smaller drones the accelerometer readings become increasingly noisy, due to less possible damping of the autopilot. Integrating accelerometers is "acceleration stable," meaning that a bias in the accelerometers that is not accounted for can lead to unbounded velocity estimates. Another option is to use the accelerometers to measure the drag on the frame, which—assuming no wind—can be easily mapped to the drone's translational velocity (cf. S. Li et al., 2018). Such a setup is "velocity stable," meaning that an accelerometer offset of drag model error would lead to a proportional velocity offset, which is bounded. On really small vehicles like the one we will use in the experiments, the accelerometers are even too noisy for reliably measuring the drag. Hence, the proposed approach uses a prediction model that only relies on the attitude estimated by the AHRS which is an indirect way of using the accelerometer. It uses the attitude and a constant altitude assumption to predict the forward acceleration, and subsequently velocity of the drone. The model is corrected from time to time by means of the visual localization. Although the IMU is used for estimating attitude, it is not used as an inertial measurement for updating translational

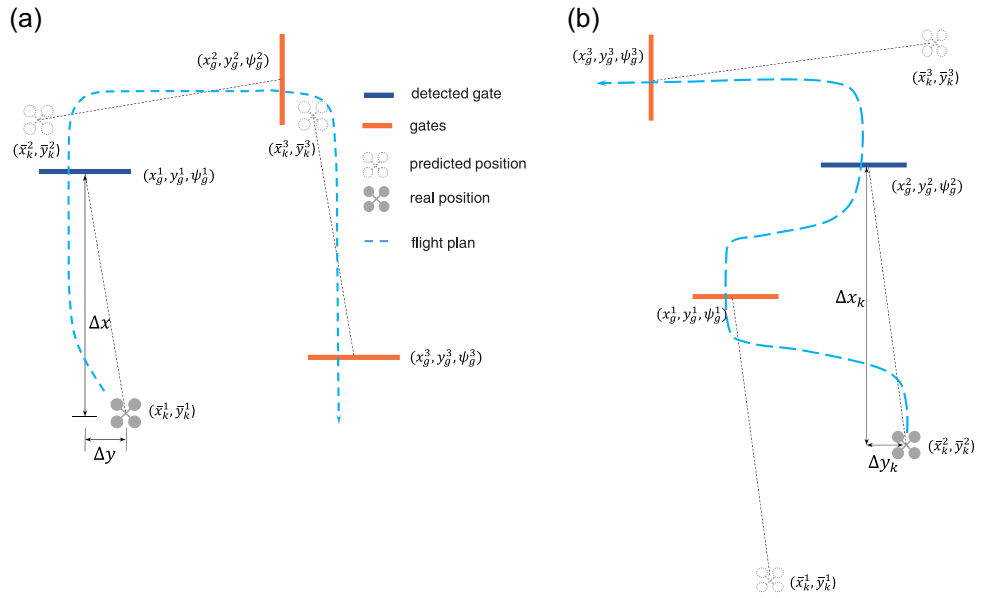

FIGURE 7 In most cases the drone will detect the next gate in the race track. However, the proposed gate assignment strategy also allows to exploit detections of other gates. (a) It iterates through all gates, evaluating where the drone would be if it was observing those gates. The position closest to the current global position is chosen as the right observation. (b) The drone detects other gate instead of the one to be flew through. This still helps state estimation, as the observed gate indeed gives an estimate closest to the current estimated global position [Color figure can be viewed at wileyonlinelibrary.com]

velocities. This leads to the name of the method; VML, which will be explained in detail in this subsection.

3.2.1 | Prediction error model

As mentioned above, the attitude estimated from the AHRS is used in the prediction of the drone's velocity and position. However, due to the AHRS bias and the model inaccuracy, the prediction will diverge from the ground truth over time. Fortunately, we have visual gate detections to provide position information. This *vision-based localization* will not integrate the error over time but it has a low frequency. Figure 8 is a sketch of what the onboard predictions and the vision measurements look like. The red curve is the prediction result diverging from the ground truth curve because of AHRS biases. The magenta dots are the low-frequency detections which distribute around the ground truth. The error between the prediction and measurements can be modeled as a linear function of time which will be explained later in this section. When the error model is estimated correctly, it can be used to compensate for the divergence of the prediction to obtain accurate state estimation.

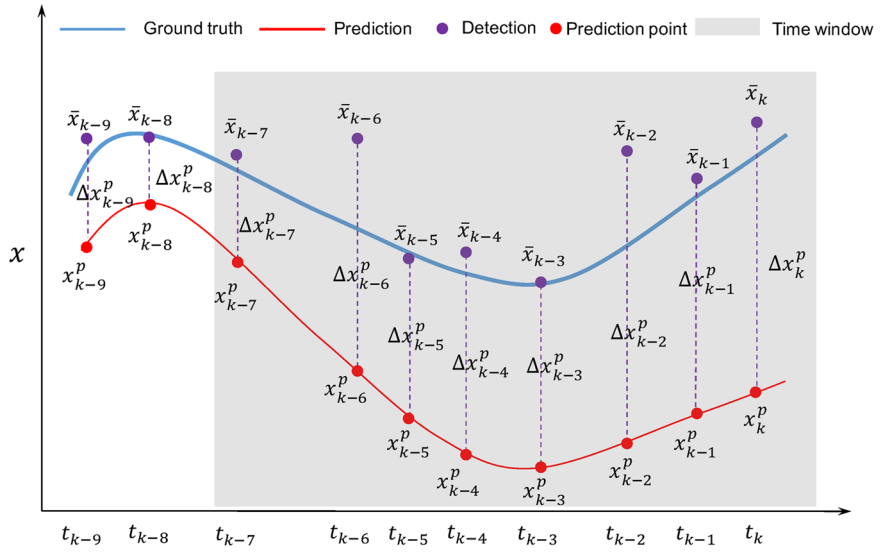

FIGURE 8 Illustrative sketch of the time window $t \in [t_{k-q}, t_k]$. At the beginning of this time window, the difference between the ground truth and the prediction is Δx_{k-q} and Δv_{k-q}. The prediction can be done with high frequency Attitude and Heading Reference System (AHRS) estimates. The vision algorithm outputs low-frequency unbiased measurements. The prediction curve deviates more and more from the ground truth curve over time because of the AHRS bias and model inaccuracy [Color figure can be viewed at wileyonlinelibrary.com]

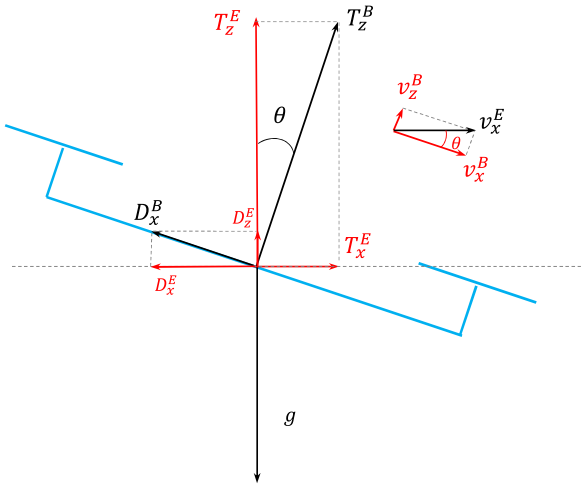

FIGURE 9 Free body diagram of the drone. $v_*^*(t)$ is the velocity of the drone. The superscript E denotes north-east-down (NED) earth frame while B denotes body frame. T_*^* is the acceleration caused by thrust and D_*^* is the acceleration caused by the drag, which is a linear function of the body velocity. g is the gravity factor and c is the drag factor which is positive. $\theta(t)$ is the pitch angle of the drone. It should be noted that since we use NED frame, $\theta < 0$ when the drone pitches down [Color figure can be viewed at wileyonlinelibrary.com]

Assuming that there is no wind, and knowing the attitude, we can predict the acceleration in the x and y axis. Figure 9 shows the forces the drone experiences. T_*^* denotes the acceleration caused by the thrust of the drone. It provides the forward acceleration together with the pitch angle θ. D_*^* denotes the acceleration caused by the drag which is simplified as a linear function of body velocity (Faessler, Franchi, & Scaramuzza, 2017):

$$\begin{cases} D_x^B = c_x v_x^B, \\ D_y^B = c_y v_y^B, \end{cases} \tag{4}$$

where c_* is the drag coefficient.

According to Newton's second law in xoz plane,

$$\begin{bmatrix} a_x(t) \\ a_z(t) \end{bmatrix} = \begin{bmatrix} 0 \\ g \end{bmatrix} + \mathfrak{R}_B^E(\theta) \begin{bmatrix} 0 \\ T_z^B(t) \end{bmatrix} + \mathfrak{R}_B^E(\theta) \mathbf{D} \mathfrak{R}_E^B(\theta) \begin{bmatrix} v_x^E(t) \\ v_z^E(t) \end{bmatrix}. \tag{5}$$

Expand Equation (5), we have

$$\begin{cases} a_x(t) = \sin\theta(t) T_z^B(t) - v_x^E(t)c, \\ a_z(t) = \cos\theta(t) T_z^B(t) + g - v_z^E(t)c, \end{cases} \tag{6}$$

where $\mathfrak{R}_E^B(\theta)$ is the rotation matrix and $\mathbf{D} = \begin{bmatrix} -c & 0 \\ 0 & -c \end{bmatrix}$ is the drag coefficient matrix. If the altitude is kept the same as in the IROS drone race, we have

$$\begin{cases} T_z^B(t) = \dfrac{-g}{\cos\theta(t)}, \\ a_x(t) = -g\tan\theta(t) - v_x^E(t)c. \end{cases} \tag{7}$$

Since the model in the y axis has the same form as in the x axis, the dynamic model of the quadrotor can be simplified as

$$\begin{cases} \dot{x}(t) &= v_x^E(t), \\ \dot{y}(t) &= v_y^E(t), \\ \dot{v}_x^E(t) &= -g\tan\theta(t) - v_x^E(t)c, \\ \dot{v}_y^E(t) &= g\tan\phi(t) - v_y^E(t)c, \end{cases} \tag{8}$$

where $x(t)$ and $y(t)$ are the position of the drone, and ϕ is the roll angle of the drone. In Equation (8), the movement in x and y axis is decoupled. Thus we only analyze the movement in the x axis. The result can be directly generalized to the y axis. The nominal model of the drone in x axis can be written by

$$\dot{\mathbf{x}}^n(t) = \mathbf{A}\mathbf{x}^n(t) + \mathbf{B}u^n(t), \tag{9}$$

where $\mathbf{x}^n(t) = \begin{bmatrix} x^n(t) \\ v_x^n(t) \end{bmatrix}$, $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & -c \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 0 \\ -g \end{bmatrix}$, and $u^n = \tan(\theta)$. The superscript n denotes the nominal model. Similarly, with the assumption that the drag factor is accurate, the prediction model can be written as

$$\dot{\mathbf{x}}^p(t) = \mathbf{A}\mathbf{x}^p(t) + \mathbf{B}u^p(t), \tag{10}$$

where $\mathbf{x}^p(t) = \begin{bmatrix} x^p(t) \\ v_x^p(t) \end{bmatrix}$ and $u^p = \tan(\theta + \theta_b)$. θ_b is the AHRS bias and is assumed to be a constant in short time. Consider a time window $t \in [t_{k-q}, t_k]$, the states of nominal model at time t_k are

$$\mathbf{x}_k^n = (\mathbf{I} + \mathbf{A}T_s)^q \mathbf{x}_{k-q}^n + \sum_{i=1}^{q} (\mathbf{I} + \mathbf{A}T_s)^{i-1} \mathbf{B}T_s u_{k-i}^n, \tag{11}$$

where T_s is the sampling time. The predicted states of model 10 are

$$\mathbf{x}_k^p = (\mathbf{I} + \mathbf{A}T_s)^q \mathbf{x}_{k-q}^p + \sum_{i=1}^{q} (\mathbf{I} + \mathbf{A}T_s)^{i-1} \mathbf{B}T_s u_{k-i}^p. \tag{12}$$

Thus, the error between the predicted model and nominal model can be written as

$$\Delta\mathbf{x}_k^p = (\mathbf{I} + \mathbf{A}T_s)^q \left[\mathbf{x}_{k-q}^p - \mathbf{x}_{k-q}^n\right] + \sum_{i=1}^{q} (\mathbf{I} + \mathbf{A}T_s)^{i-1} T_s \mathbf{B}u_b, \tag{13}$$

where $u_b = \left(u_{k-i}^p - u_{k-i}^n\right)$ is the input bias which can be considered as a constant in a short time. In Equation (13),

$$(\mathbf{I} + \mathbf{A}T_s)^i = \begin{bmatrix} 1 & T_s \sum_{j=1}^{i} (1 - cT_s)^{j-1} \\ 0 & (1 - cT_s)^i \end{bmatrix}. \tag{14}$$

Since the sampling time T_s is small, ($T_s = 0.002$ s in our case), we can assume

$$(\mathbf{I} + \mathbf{A}T_s)^i \approx \begin{bmatrix} 1 & iT_s \\ 0 & 1 \end{bmatrix}. \tag{15}$$

Hence, Equation (13) can be approximated by

$$\Delta \mathbf{x}_k^p = (\mathbf{I} + \mathbf{A}T_s)^q \left[\mathbf{x}_{k-q}^p - \mathbf{x}_{k-q}^n\right] + \sum_{i=1}^{q} \begin{bmatrix} 1 & iT_s \\ 0 & 1 \end{bmatrix} T_s \mathbf{B} u_b, \tag{16}$$

$$= \begin{bmatrix} 1 & qT_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x_k^p \\ \Delta v_k^p \end{bmatrix} + \begin{bmatrix} q & \dfrac{q(q+1)}{2} T_s \\ 0 & q \end{bmatrix} T_s \mathbf{B} u_b. \tag{17}$$

Expanding Equation (17), we have

$$\begin{cases} \Delta x_k^p = \Delta x_{k-q}^p + qT_s \Delta v_{xk-q}^p - \dfrac{q(q+1)}{2} T_s^2 g u_b, \\ \Delta v_k^p = \Delta v_{xk-q}^p - qT_s g u_b. \end{cases} \tag{18}$$

Actually, $qT_s = t_k - t_{k-q}$ is the time span of the time window. If we neglect T_s^2 term, we can have the prediction error at time t_k

$$\Delta x_k^p = \Delta x_{k-q}^p + (t_k - t_{k-q}) \Delta v_{k-q}^p. \tag{19}$$

Thus, within a time window, the state estimation problem can be transformed to a linear regression problem with model Equation (19), where $\hat{\boldsymbol{\beta}} = \left[\Delta x_{k-q}^p, \Delta v_{k-q}^p\right]^{\mathrm{T}}$ are the parameters to be estimated. From Equation (19), we can see that in a short time window, the AHRS bias does not affect the prediction error. The error is mainly caused by the initial prediction error Δx_{k-q}^p. Furthermore, velocity error Δv_{k-q}^p can cause the prediction error to diverge over time. If the time window is updated frequently, model 19 can remain accurate enough. Hence, in this study, we focus on the main contributors to the prediction error and will not estimate the bias term. The next step is how to efficiently and robustly estimate Δx_{k-q}^p and Δv_{k-q}^p.

In this simplified linear prediction error model, we use the constant altitude assumption to approximate the thrust T_z^B on the drone, which may lead to inaccuracy of the model. During the flight, this assumption may be violated by aggressive maneuvers in z axis. However, if the maneuver in z axis is not very aggressive and the time window is small (in our case less than 2 s), the prediction error model's inaccuracy level can be kept in an acceptable range. In the simulation and the real-world experiment shown later, we will show that although the altitude of the drone changes 1 m in 2 s, the proposed filter can still have very high accuracy with this assumption. Another way to improve the model accuracy is to estimate the thrust by fusing the accelerometer readings and rotor speed together, which needs the establishment of the rotors' model. It should also be noted that we neglect T_s^2 term in Equation (18) to have a linear model. To increase the model accuracy, the prediction error model can be a quadratic model. In our case, since the time window is small, the linear model is accurate enough.

3.2.2 | Parameter estimation method

The classic way for solving the linear regression problem based on Equation (19) is to use the least square method (LS Method) with all data within the time window and estimate the parameters $\hat{\boldsymbol{\beta}}$.

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y}, \tag{20}$$

where

$$\hat{\boldsymbol{\beta}} = \left[\Delta x_{k-q}^p \quad \Delta v_{k-q}^p\right], \quad \mathbf{X} = \begin{bmatrix} 1 & t_{k-q} - t_{k-q} \\ 1 & t_{k-q+1} - t_{k-q} \\ \vdots & \vdots \\ 1 & t_k - t_{k-q} \end{bmatrix},$$

$$\mathbf{Y} = \begin{bmatrix} x_{k-q}^p - \bar{x}_{k-q} \\ x_{k-q+1}^p - \bar{x}_{k-q+1} \\ \vdots \\ x_k^p - \bar{x}_k \end{bmatrix}.$$

The LS Method in Equation (20) can give optimal unbiased estimation. However, if there exist outliers in the time window $t \in [t_{k-q}, t_k]$, they will be considered equally during the estimation process. These outliers can significantly affect the estimation result. Thus, to exclude the outliers, we employ random sample consensus (RANSAC) to increase the performance (Fischler & Bolles, 1981). In a time window $t \in [t_{k-q}, t_k]$, we first calculate the prediction error $\Delta \mathbf{x}_{k-q,k}^p = \left\{\Delta x_{k-q+i}^p | \Delta x_{k-q+i}^p = x_{k-q+i}^p - \bar{x}_{k-q+i}, 0 \le i \le q\right\}$ and time difference $\Delta \mathbf{t} = \{\Delta t_i | \Delta t_i = t_i - t_{k-q}, 0 \le i \le q\}$. For each iteration i, the subsets of $\Delta \mathbf{x}_{k-q,k}^p$ and $\Delta \mathbf{t}_{k-q,k}$ are randomly selected, which are denoted by $\Delta \mathbf{x}_{k-q,k}^s$ and $\Delta \mathbf{t}_{k-q,k}^s$. The size of the subset n^s can be calculated by $n^s = q\sigma_s$, where σ_s is the ratio of sampling. We use subsets $\Delta \mathbf{x}_{k-q,k}^s$ and $\Delta \mathbf{t}_{k-q,k}^s$ to estimate the parameters $\hat{\boldsymbol{\beta}}_i$ (Figure 10).

When $\hat{\boldsymbol{\beta}}_i$ is estimated, it will be used to calculate the total prediction error ε_i of the all the data in the time window $t_i \in [t_{k-q}, t_k]$ by

$$\varepsilon_i = \sum_{j=k-q}^{k} \epsilon_j, \tag{21}$$

where

$$\epsilon_j = \begin{cases} \| \Delta v_{k-q_i}^p (\Delta t_j - \Delta t_{k-q}) + \Delta x_{k-q_i}^p - \Delta x_j^p \|_2, & \text{if} \quad \epsilon_j < \sigma_{th}, \\ \sigma_{th}, & \text{otherwise.} \end{cases} \tag{22}$$

In the process of Equation (21), if ϵ_j is larger than a threshold σ_{th}, it counts the threshold as the error. After all the iterations, the parameters $\hat{\boldsymbol{\beta}}_i$ which has the least prediction error will be selected to be the estimated parameters for this time window $t_i \in [t_{k-q}, t_k]$. The pseudo-code of this Basic RANSAC Fitting (BRF) method can be found in Algorithm 2.

With the BRF method, the influence of the outliers is reduced, but it has no mechanism to handle over-fitting. For example, in time window $t_i \in [t_{k-q}, t_k]$, BRF can estimate the optimal parameters $\hat{\beta}$ with the minimal error. However, sometimes it will set Δv_{k-q}^p to unrealistically high values. This happens when there are few detections in the time window, which may result in the inaccurate estimation of the parameters. In reality, the drone flies at maximum speed 3 m/s, so the velocity prediction error at the start of time window t_{k-q} should not be too large. To avoid over-fitting, we add a penalty factor/prior matrix $\mathbf{P}$ to limit Δv_{k-q}^p in the fitting process. The loss function can be written as

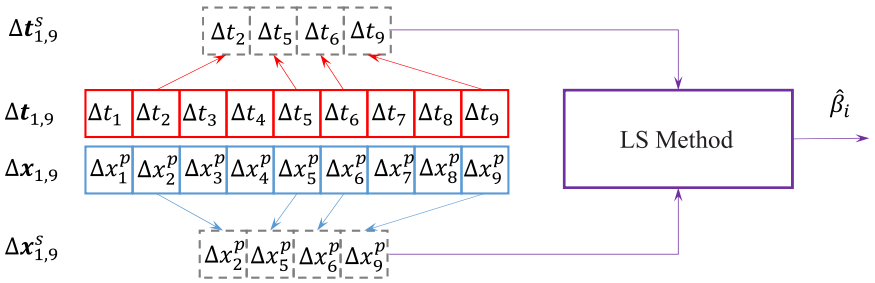

FIGURE 10 In the ith iteration, the data in the time window $t \in [t_1, t_9]$ will be randomly sampled into $\Delta \mathbf{t}^s_{k-q,k}$ and $\Delta \mathbf{x}^s_{k-q,k}$. Then least square method (LS Method; Equation 20) will be used to estimate the parameters $\hat{\boldsymbol{\beta}}_i$. In this example, $\sigma_s = 0.4$, which means that $n^s = 9 \times 0.4 \approx 4$ samples should be sampled [Color figure can be viewed at wileyonlinelibrary.com]

$$J(\hat{\boldsymbol{\beta}}) = \|\mathbf{X}\hat{\boldsymbol{\beta}} - \mathbf{Y}\|_2^2 + \hat{\boldsymbol{\beta}}^{\mathsf{T}}\mathbf{P}\hat{\boldsymbol{\beta}}, \tag{23}$$

where

$$\mathbf{P} = \begin{bmatrix} p_x & 0 \\ 0 & p_v \end{bmatrix} \tag{24}$$

is the penalty factor/prior matrix. To minimize the loss function, we take derivatives of $J(\hat{\boldsymbol{\beta}})$ and let it be 0

$$\frac{\partial J(\hat{\boldsymbol{\beta}})}{\partial \hat{\boldsymbol{\beta}}} = 2\mathbf{X}^{\mathsf{T}}\mathbf{X}\hat{\boldsymbol{\beta}} - 2\mathbf{X}^{\mathsf{T}}\mathbf{Y} + \mathbf{P}\hat{\boldsymbol{\beta}} + \mathbf{P}^{\mathsf{T}}\hat{\boldsymbol{\beta}} = 0. \tag{25}$$

Then we have the estimated parameters by

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathsf{T}}\mathbf{X} + \mathbf{P})^{-1}\mathbf{X}^{\mathsf{T}}\mathbf{Y}. \tag{26}$$

We call the use of Equation (26) inside the RANSAC fitting the Prior RANSAC fitting (PRF). Compared with Equation (20), PRF has the penalty factor/prior matrix $\mathbf{P}$ in it. By tuning matrix $\mathbf{P}$ we can add the prior knowledge to the parameter distribution. For example, in our case Δv^p_{k-q} should not be high. Thus, we can increase p_v in $\mathbf{P}$ to limit the value of Δv^p_{k-q}.

To conclude, in this part we propose three methods for estimating the parameters $\hat{\boldsymbol{\beta}}$. The first one is the LS Method which considers all the data in a time window equally. The second method is BRF, which has the mechanism to exclude the outliers. And the third one is PRF, which can not only exclude the outliers but also take into account the prior knowledge to avoid overfitting. In the next section, we will discuss and compare these three methods in simulation to see which one is the most suitable for our drone race scenario.

3.2.3 | Prediction compensation

After the error model (Equation 19) is estimated in time window k, the error model can be used to compensate the prediction by

$$\begin{bmatrix} \hat{x}_{k+i} \\ \hat{v}_{k+i} \end{bmatrix} = \begin{bmatrix} x^p_{k+i} \\ v^p_{k+i} \end{bmatrix} - \begin{bmatrix} 1 & t_{k+i} - t_{k-q} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x^p_{k-q} \\ \Delta v^p_{k-q} \end{bmatrix}. \tag{27}$$

Also, at each prediction step, the length $\Delta T = t_k - t_{k-q}$ of the time window will be checked, since the simplified model 19 is based on the assumption that the time span of the time window ΔT is small. If ΔT is larger than the allowed maximum time window size $\Delta T_{\max}$, the filter will delete the oldest elements until $\Delta T < \Delta T_{\max}$. The pseudo-code of the proposed VML with LS Method can be found in Algorithms 3 and 4.

3.2.4 | Comparison with Kalman filter

When it comes to state estimation or filtering technique, it is inevitable to mention the Kalman filter which is the most commonly used state estimation method. The basic idea of the EKF is that at time t_{k-1}, it first predicts the states at time t_k with its error covariance $\mathbf{P}_{k|k-1}$ to have prior knowledge of the states at t_k.

$$\begin{aligned} \hat{\mathbf{X}}_{k|k-1} &= \hat{\mathbf{X}}_{k-1} + \mathbf{f}(\hat{\mathbf{X}}_{k-1}, \mathbf{u}_{k-1})\mathsf{T}_s, \\ \mathbf{F}_{k-1} &= \frac{\partial}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))|_{\mathbf{x}(t)=\hat{\mathbf{x}}_{k-1}}, \\ \Phi_{k|k-1} &\approx \mathbf{I} + \mathbf{F}_{k-1}\mathsf{T}, \\ \mathbf{H}_k &= \frac{\partial}{\partial \mathbf{x}}\mathbf{h}(\mathbf{x}(t))|_{\mathbf{x}(t)=\hat{\mathbf{x}}_k}, \\ \mathbf{P}_{k|k-1} &= \Phi_{k|k-1}\mathbf{P}_{k-1}\Phi^{\mathsf{T}}_{k|k-1} + \mathbf{Q}_{k-1}. \end{aligned} \tag{28}$$

When an observation arrives, the Kalman filter uses an optimal gain $\mathbf{K}_k$ which is a combination of the prior error covariance $\mathbf{P}_{k+1|k}$ and the observation's covariance $\mathbf{R}_k$ to compensate the prediction, which as a result, leads to the minimum error covariance $\mathbf{P}_k$.

$$\begin{aligned} \delta\hat{\mathbf{X}}_k &= \mathbf{K}_k\{\mathbf{Z}_k - \mathbf{h}[\hat{\mathbf{X}}_{k|k-1}, k]\}, \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1}\mathbf{H}^{\mathsf{T}}_k\left[\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}^{\mathsf{T}}_k + \mathbf{R}_k\right]^{-1}, \\ \hat{\mathbf{X}}_k &= \hat{\mathbf{X}}_{k|k-1} + \delta\hat{\mathbf{X}}_k, \\ \mathbf{P}_k &= (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k/k-1}(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^{\mathsf{T}} + \mathbf{K}_k\mathbf{R}_k\mathbf{K}^{\mathsf{T}}_k. \end{aligned} \tag{29}$$

According to Diderrich (1985), a Kalman filter is a least square estimation made into a recursive process by combining prior data with coming measurement data. The most obvious difference between the Kalman filter and the proposed VML is that VML is not a recursive method. It does not estimate the states at t_k only based on the last step states $\hat{\mathbf{x}}_{k-1}$. It estimates the states considering the previous prediction and observations in a time window.

In the VML approach, we use least square method within a time window, which looks similar to the least square estimation method.

However, there are two major differences between the two methods. The first one is that in the proposed VML, the prediction information is fused to the VML. Secondly and most importantly, we estimate the prediction error model $\hat{\beta}$ instead of estimating all the states in the time window as in the least square method. Thus, the VML has its advantages of handling outliers and delay by its time window mechanism and it also has the advantage of computational efficiency to the Least Square Estimation. In Section 4, we will introduce Kalman filter's different variants for outliers and delay and compare them with VML in estimation accuracy and computation load in detail.

3.3 | Flight plan and high-level control

With the state estimation method explained above, to fly a racing track, we employ a flight plan module which sets the waypoints that guide the drone through the track and a two-loop cascade P-controller to execute the reference trajectory (Figure 11).

Usually, the waypoint is just behind the gate. When the distance between the drone and the waypoint is less than a threshold D_{turn}, the gate can no longer be detected by our method, and we set the heading of the drone to the next waypoint. This way, the drone will start turning towards the next gate before arriving at the waypoint. When the distance between the drone and the waypoint is within another threshold D_{switch_wp}, the waypoint switches to the next point. With this strategy, the drone will not stop at one waypoint but already start accelerating to the next waypoint, which can help to save time. The work flow of flight plan module can be found in Algorithm 5.

We employ a two-loop cascade P-controller (Equation 30) to control the drone to reach the waypoints and follow the heading reference generated from the flight plan module. The altitude and attitude controllers are provided by the Paparazzi autopilot, and are both two-loop cascade controllers.

$$\mathbf{\Phi}^c(k) = \mathbf{R}_\psi \mathbf{K}_v(\mathbf{K}_x(\mathbf{x}^r(k) - \hat{\mathbf{x}}(k)) - \hat{\mathbf{v}}(k)), \tag{30}$$

where

$$\mathbf{\Phi}^c(k) = [\theta^c(k), \phi^c(k)]^\mathbb{T}, \mathbf{R}_\psi = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix}, \mathbf{K}_v = \begin{bmatrix} k_{v_x} & 0 \\ 0 & k_{v_y} \end{bmatrix}, \mathbf{K}_x = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix}, \mathbf{x}^r(k) = [x^r(k), y^r(k)]^\mathbb{T}, \hat{\mathbf{x}}(k) = [\hat{x}(k), \hat{y}(k)]^\mathbb{T}, \hat{\mathbf{v}}(k) = [\hat{v}_x(k), \hat{v}_y(k)]^\mathbb{T}.$$

4 | SIMULATION EXPERIMENTS

4.1 | Simulation setup

To verify the performance of VML in the drone race scenario, we first test it in simulation and then use an EKF as benchmark to compare both filters to see which one is more suitable in different operation points. We first introduce the drone's dynamics model used in the simulation.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix},$$
$$\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \mathfrak{R}_B^E \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} + \mathfrak{R}_B^E \mathbf{K} \mathfrak{R}_E^B \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}, \tag{31}$$
$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{T} \end{bmatrix} = \begin{bmatrix} k_\phi(\phi^c - \phi) \\ k_\theta(\theta^c - \theta) \\ k_\psi(\psi^c - \psi) \\ k_T(T^c - T) \end{bmatrix},$$

where (x, y, z) is the position of the drone in the Earth frame. v_* is the velocity of the drone. g is the gravity factor. T is the acceleration caused by the thrust force. ϕ, θ, ψ are the three Euler angles of the body frame. And $\mathfrak{R}_B^E$ is the rotation matrix from the Body frame to the Earth frame. $\mathbf{K} = \text{diag}([-0.5, -0.5, 0])$ is the simplified first order drag matrix, where the values are based on a linear fit of the drag based on real-world data with the Trashcan drone. $\mathfrak{R}_B^E \mathbf{K} \mathfrak{R}_E^B [v_x v_y v_z]^\mathbb{T}$ is the acceleration caused by other aerodynamics. The last four equations are the simplified first order model of the attitude

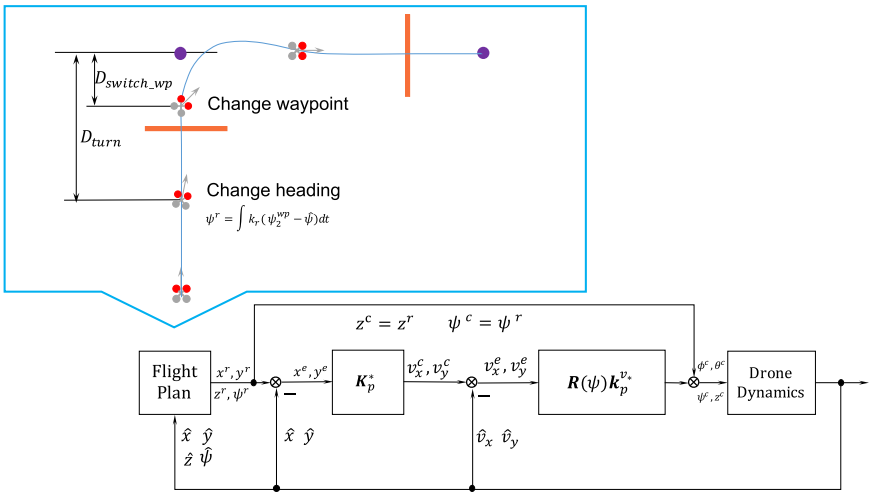

FIGURE 11 The Flight plan module generates the waypoints for the drone to fly the track. When the distance between the drone and the current waypoint $d < D_{turn}$, the drone starts to turn to the next waypoint while still approaching the current waypoint. When $d < D_{switch\_wp}$, the drone switches the current waypoint to the next one. The cascade P-controller is used for executing the reference trajectory from the flight plan module. The attitude and rate controllers are provided by the Paparazzi autopilot. k_r is a positive constant to adjust the speed of the drone's yawing to the setpoint. In the real-world experiment and simulation, we set $k_r = 1$ [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 2 The map of the simulated racing track

Gate ID	x (m)	y (m)	z (m)	ψ [°]
1	4	0	−1.5	0
2	4	4	−2.5	90
3	0	4	−1.0	180
4	0	0	−1.5	270

controller and thrust controllers where the proportional feedback factors are $k_\phi = 6, k_\theta = 6, k_\psi = 5, k_T = 3$. Thus, the model 31 in the simulation is a 10 states $\mathbf{x} = [x, y, z, v_x, v_x, v_x, \phi, \theta, \psi, T]^\mathbb{T}$ and four inputs $\mathbf{u} = [\phi^c, \theta^c, \psi^c, T^c]^\mathbb{T}$ nonlinear system. In this simulation, we use the same flight plan module and high-level controllers discussed in Section 3 (Figure 11) to generate a ground truth trajectory through a 4-gate square racing track (Table 2). In this track, we use different height to test if the altitude change affects the accuracy of the VML.

With the ground truth states, next step is to generate the sensor reading. In the real world, AHRS estimation outputs biased attitude estimation because of the accelerator's bias. To model AHRS bias, we have a simplified AHRS bias model

$$\begin{bmatrix}\phi_b\\ \theta_b\end{bmatrix} = \begin{bmatrix}\cos\psi & \sin\psi\\ -\sin\psi & \cos\psi\end{bmatrix}\begin{bmatrix}B_N\\ B_E\end{bmatrix}, \tag{32}$$

where ϕ_b and θ_b are the AHRS biases on ϕ and θ. B_N and B_E are the north and east bias caused by the accelerometer bias, which can be considered as constants in short time. From real-world experiments, they are less than 3°. Thus, the AHRS reading can be modelled by

$$\begin{bmatrix}\bar{\phi}_k\\ \bar{\theta}_k\end{bmatrix} = \begin{bmatrix}\phi_k\\ \theta_k\end{bmatrix} + \begin{bmatrix}\cos\psi & \sin\psi\\ -\sin\psi & \cos\psi\end{bmatrix}\begin{bmatrix}B_N\\ B_E\end{bmatrix} + \begin{bmatrix}\epsilon_\phi\\ \epsilon_\theta\end{bmatrix}, \tag{33}$$

where ϵ_*~$N(0, \sigma_*)$ is the AHRS noise and in our simulation we will set $\sigma_* = 0.5°, B_N = -2°, B_E = 1°$. For vision measurements generation, we first determine the segment $[u, v]$ of the trajectory where the drone can detect the gate. Then, we calculate the number of the detection by $n_v = \frac{t_u - t_v}{f_v}$, where f_v is the detection frequency. Next, we randomly select n_v points between u and v to be vision points. For these points, we generate detection measurement by

$$\begin{bmatrix}\bar{x}_k\\ \bar{y}_k\end{bmatrix} = \begin{bmatrix}x_k\\ y_k\end{bmatrix} + \begin{bmatrix}\epsilon_x\\ \epsilon_y\end{bmatrix}. \tag{34}$$

In Equation (34), ϵ_*~$N(0, \sigma_*)$ is the detection noise and $\sigma_* = 0.1\,\text{m}$ In these n_v vision points, we also randomly select a few points as outlier points, which have the same model with Equation (34) but $\sigma_* = 3\,\text{m}$. In the following simulations, the parameters are the same with the value mentioned in this section if there is no statement. The simulated ground truth states and sensor measurements are shown in Figure 12.

4.2 | Simulation result and analysis

4.2.1 | Comparison between EKF, BRF, and PRF without outliers

We employ an EKF as benchmarks to compare the performance of our proposed filter. The details of the EKF can be found in the Appendix. We first do the simulation in only one operation point, where $f_v = 30\,\text{m}$ HZ, $\sigma_* = 0.1\,\text{m}$ and the probability of outliers

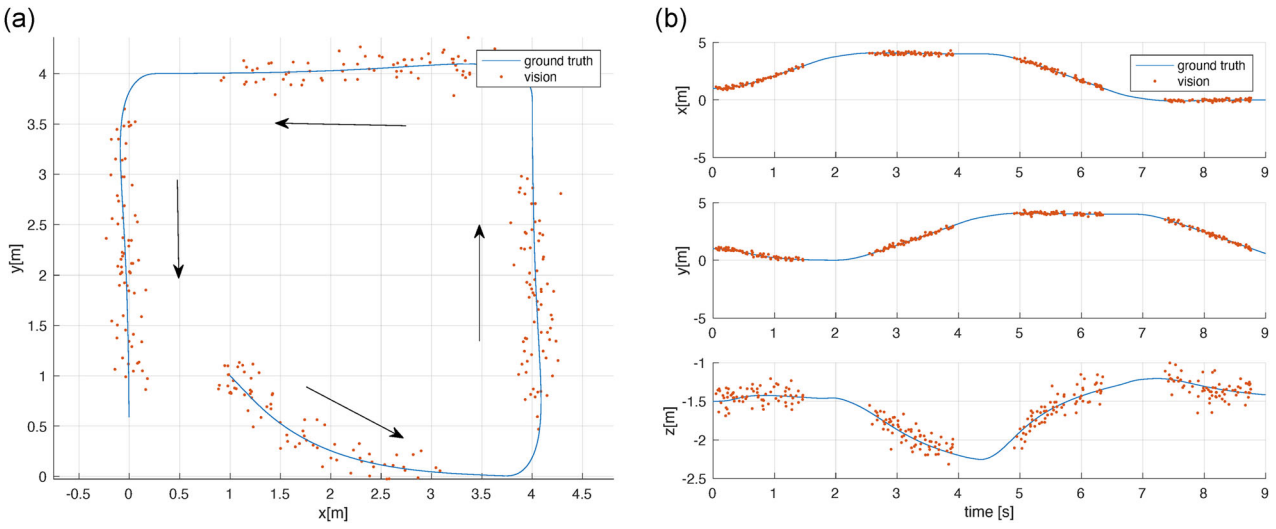

FIGURE 12 In the simulation, the ground truth states are first generated (blue curve). Then, vision measurements and Attitude and Heading Reference System (AHRS) readings are generated. It can be seen clearly that the bias of AHRS readings changes with the heading, as on a real drone. Namely, the offset in ϕ and θ changes when the ψ changes. This phenomenon is modeled by Equation (32). In this simulation $f_v = 30$ Hz, $\sigma_x = \sigma_y = \sigma_z = 0.1\,\text{m}$ (a) Generated ground truth states and vision measurements in $x - y$ plane. (b) Generated ground truth position and vision measurements [Color figure can be viewed at wileyonlinelibrary.com]

$P_{out} = N_{outliers}/N_{detection} = 0$. At this operation point, three filters are run separately. The result is shown in Figure 13.

When there are no outliers, all three filters can converge to the ground truth value. However, the EKF has a longer startup period and BRF overfits after turning, leading to unlikely high velocity offsets (the peaks in Figure 13b). This is because, after the turn, the RANSAC buffer is empty. When the first few detections come into the buffer, the RANSAC has a larger chance to estimate inaccurate parameters. In PRF, however, we add a prior matrix $\mathbf{P} = \begin{bmatrix} 0 & 0 \\ 0 & 0.3 \end{bmatrix}$ to limit the value of Δv and the number of the peaks in the velocity estimation is significantly decreased. At the same time, the velocity estimation is closer to the ground truth value.

To evaluate the estimation accuracy of each filter, we first introduce a variable, average estimation error γ, to be an index of the filter's performance:

$$\gamma = \sqrt{\frac{\sum_{i=1}^{N}(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2}{N}}, \tag{35}$$

where N is the number of the sample points on the whole trajectory. $\hat{x}$ and $\hat{y}$ are the estimated states by the filter. x and y are the ground truth positions generated by the simulation. γ captures how much the estimated states deviate from the ground truth states. A smaller γ indicates a better filtering result.

We use running time to evaluate the computation efficiency of each filter. It should be noted that since we need to store all the simulation data for visualization and MATLAB has no mechanism of passing pointers, data accessing can take much computation time. Thus, we only count the running time of the core parts of the filters, which are the prediction and the correction.

The results are shown in Figure 14. In the simulation, the time window in BRF and PRF is set to be 1 s and five iterations are performed in the RANSAC procedure. For each frequency, the filters are run 10 times separately and their average γ and running time are calculated. It can be seen in Figure 14a that when the detection frequency is larger than 30 Hz, BRF and PRF perform close to the EKF. In terms of calculation time, the EKF is heavier than BRF and PRF when the frequency is lower than 40 Hz. It is because that during the prediction phase, the EKF not only predicts the states but also calculates the Jacobian matrix and the prior error covariance $\mathbf{P}_{k|k-1}$ by high frequency while BRF and PRF only do the state prediction. However, when the detection comes, the EKF does the correction by several matrix operations while BRF and PRF do the RANSAC which is much heavier. This explains why the EKF's computation load is only slightly affected by the detection frequency but BRF and PRF's computation load increases significantly with higher detection frequency.

4.2.2 | Comparison between EKF, BRF, and PRF with outliers

When outliers appear, the regular EKF can be affected significantly. Thus, outlier rejection strategies are always used within an EKF to increase its robustness. A commonly used method is using Mahalanobis distance between the observation and its mean as an index to determine whether an observation is an outlier (Chang, 2014; Z. Li, Chang, Gao, Wang, & Hernandez, 2016). Thus, in this section, we implement an EKF with outlier rejection (EKF-OR) as a benchmark to compare the outlier rejection performance of BRF and PRF. The basic idea for the EKF-OR is that the square of the observation's Mahalanobis distance is

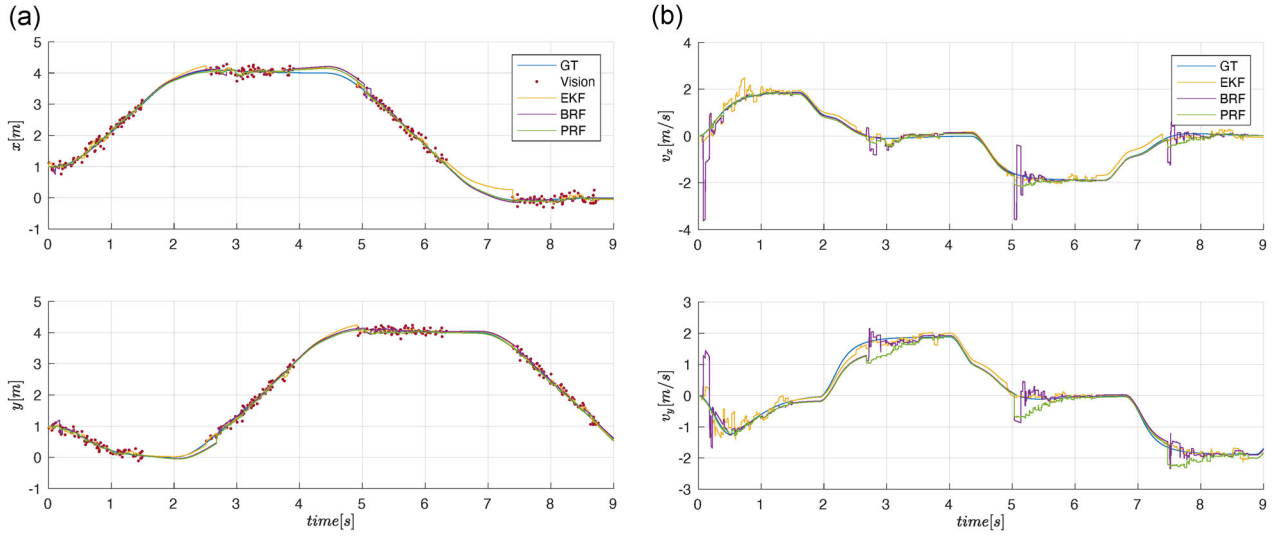

FIGURE 13 The filtering result of EKF, Basic RANSAC Fitting (BRF), and Prior RANSAC fitting (PRF). $f_v = 50$ Hz and $\sigma_x = \sigma_y = 0.1$. When there are no outliers, EKF, BRF, and PRF's estimating result all converge to ground truth value. In velocity estimation, however, EKF has longer startup period than VML and BRF shows peaks, which is caused by the over-fitting. To limit this over-fitting, in PFR, we add a prior matrix $\mathbf{P} = \begin{bmatrix} 0 & 0 \\ 0 & 0.3 \end{bmatrix}$ and the velocity's peak is significantly smoothed and is closer to the ground truth velocity. (a) Position estimation of EKF, BRF and PRF. (b) Velocity estimation of EKF, BRF, and PRF [Color figure can be viewed at wileyonlinelibrary.com]

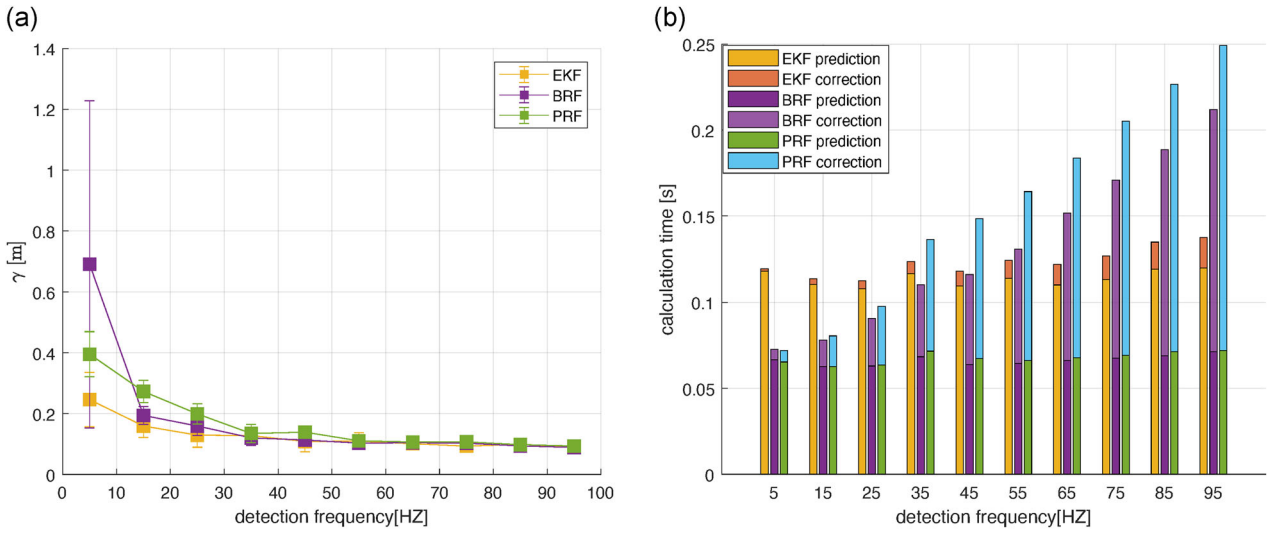

FIGURE 14 The simulation result of the filters. It can be seen that when the detection frequencies are below 20 Hz, the EKF performs better than Basic RANSAC Fitting (BRF) and Prior RANSAC fitting (PRF). However, when the detection frequencies are higher than 20 Hz, BRF and PRF start performing better than the EKF. In terms of computation time, the EKF is affected by the detection frequency slightly while the computation load of BRF and PRF increase significantly higher detection frequencies. (a) Estimation error of the filters with different detection frequencies. (b) Calculation time of the filters [Color figure can be viewed at wileyonlinelibrary.com]

χ^2 distributed. Hence, when the observation arrives, its Mahalanobis distance will be calculated and checked whether it is within a threshold χ_α. If it is not, this observation will be rejected.

Two examples of the filters' rejecting outliers are shown in Figure 15. The first figure shows a common case that the three filters can reject the outliers successfully. However, in some special cases, EKF-OR is vulnerable to the outliers. In Figure 15b, for instance, after a long time of pure prediction, the error covariance $\mathbf{P}_{k|k-1}$ becomes large. Once EKF-OR meets an outlier, it has a high chance to jump to it. The subsequent true positive detections will be treated as outliers and EKF-OR starts diverging. At the same time, BRF and PRF are more robust to the outliers. The essential reason is that for EKF-OR, it depends on its current state estimation (mean and error covariance) to identify the outliers. When the current state estimation is not accurate enough, like the long-time prediction in our case, EKF-OR loses its ability to identify outliers. In other words, it tends to trust whatever it meets. The worse situation is that after jumping to the outlier, its error covariance become smaller which, as a

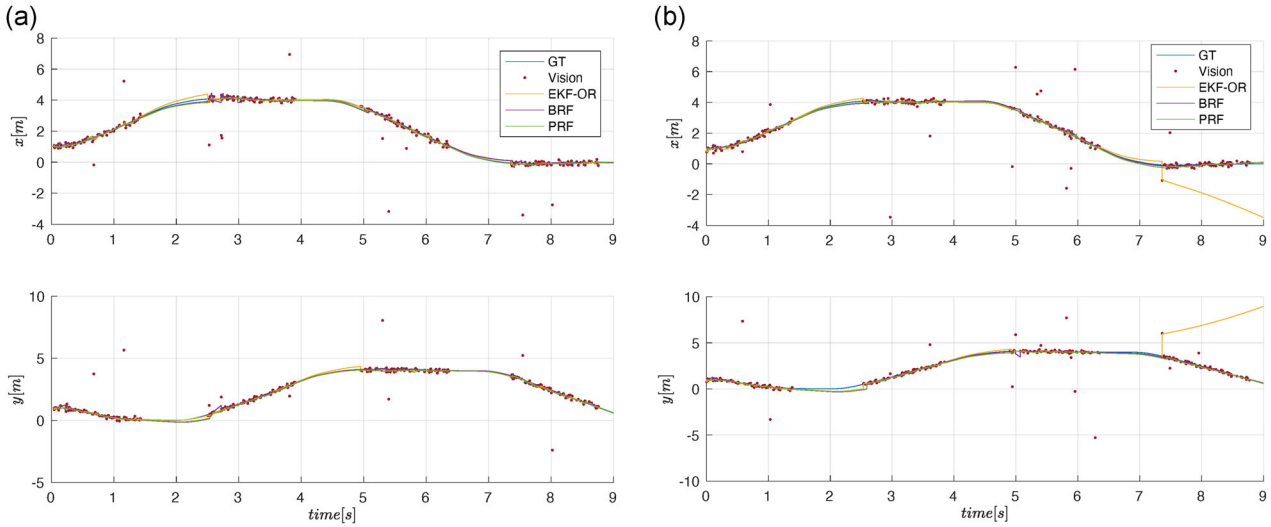

FIGURE 15 In most cases, EKF with outlier rejection (EKF-OR), Basic RANSAC Fitting (BRF), and Prior RANSAC fitting (PRF) can reject the outliers. But after a long time of pure prediction, EKF-OR is very vulnerable to the outliers while BRF and PRF still perform well. (a) When outliers appear, EKF-OR, BRF, and PRF can reject them. (b) After a long time of pure prediction, EKF-OR has large error covariance. Once it meets an outlier, it has a high chance to jump to it. As a consequence, the later true positive detections are beyond the threshold χ_α and EKF-OR will treat them as outliers [Color figure can be viewed at wileyonlinelibrary.com]

consequence, leads to the rejection of the coming true positive detections. However, for BRF and PRF, outliers are determined in a time window including history. Thus, after long time of prediction, when BRF and PRF meet an outlier, they will judge it considering the detections in the past. If there is no other detection in the time window, they will wait for enough detections to make a decision. With this mechanism, BRF and PRF become more robust than EKF-OR especially when EKF-OR's estimation is not accurate.

Figure 16 shows the estimation error and the calculation time of the three filters. As we stated before, although EKF-OR has the mechanism of dealing with the outliers, it still can diverge due to the outliers in some special cases. Thus, in Figure 16a EKF-OR has large estimation error when the detection frequency is both low and high. In terms of calculation time, it can be seen that it has no significant difference with the non-outlier case.

4.2.3 | Filtering result with delayed detection

Image processing and visual algorithms can be very computationally expensive for running onboard a drone, which can lead to significant delay (van Horssen, van Hooijdonk, Antunes, & Heemels, 2019; Weiss et al., 2012). Many visual navigation approaches ignore this delay and directly fuse the visual measurements with the onboard sensors, which sacrifices the accuracy of the state estimation. A commonly used approach for compensating this vision delay is a modified Kalman filter proposed by Weiss et al. (2012). The main idea of this approach, called EKF delay handler (EKF-DH), is having a buffer to store all sensor measurements within a certain time. At time t_k, a vision measurement corresponding to the states at earlier time t_s arrives. It will be used to correct the states at time t_s. Then, the states will be propagated again from t_s to t_k (Figure 17a). Although updating the covariance matrix is not needed according to Weiss et al. (2012), this approach still requires updating history states whenever a measurement arrives, which can be computationally expensive especially when the delay and the measurement frequency get larger. In our case, we need to use the error covariance for outlier rejections, it is necessary to update the history error covariance matrices, which in turn increases the computation load further. At the same time, for VML, when the measurement arrives, it will first be pushed into the buffer. Then, the error model will be estimated within the buffer/time window. With the estimated parameter $\hat{\beta}$, the prediction at t_k can be corrected directly without the need of correcting all the states between t_s and t_k (Figure 17b). Thus, the computational burden will not increase when the delay exists.

Figure 18 shows an example of the simulation result of the three filters when both outliers and delay exist. In this simulation, the visual delay is set to be 0.1 s. It can be seen that although there is a lag between the vision measurements and the ground truth, all the filters can estimate accurate states. However, EKF-DH requires much more computation effort. Figure 19 shows the estimation error and the computation time of the three filters.

In Figure 19, we can see that the computation load of EKF-DH increases significantly due to its mechanism of handling delay. Unsurprisingly, EKF-DH is still sensitive to some outliers while BRF and PRF can handle the outliers.

5 | REAL-WORLD EXPERIMENTS (Figure 20)

5.1 | Processing time of each component

Before testing the whole system, we first test on the ground how much time the Snake gate detection, the VML-prediction, the VML-correction and the controller take when running on a Jevois smart camera. On the ground, we set an orange gate in front of a Jevois camera and calculate the time that each component takes. For each image, for example, we start timing when a new image arrives and the Snake gate detection is run. Then, we stop timing when the snake gate finishes. For the VML and the controller, we use the same strategy to calculate processing time. In this test, the vision detection frequency is 15 Hz and the number of RANSAC iterations in VML is set to 5. Figure 21 shows the timing statistics for each component on the Jevois. It can be seen that outliers happen during the testing

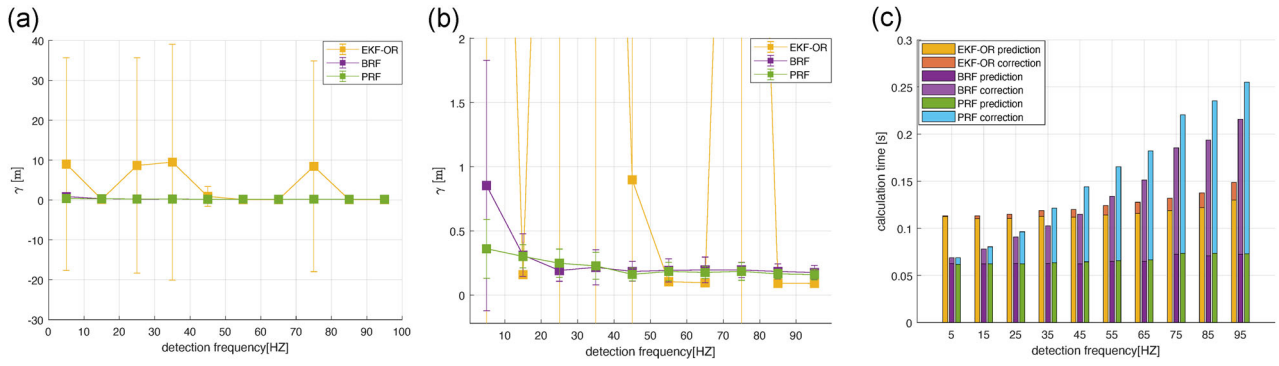

FIGURE 16 The estimation error of EKF with outlier rejection (EKF-OR), Basic RANSAC Fitting (BRF), and Prior RANSAC fitting (PRF) and their calculation time with outliers. EKF-OR has some chance (15%) to diverge, which leads to the high estimation error. (a) Estimation error of the EKF-OR, BRF, and PRF with different detection frequencies. (b) Partial enlarged drawing of (a). (c) Calculation time of the filters [Color figure can be viewed at wileyonlinelibrary.com]

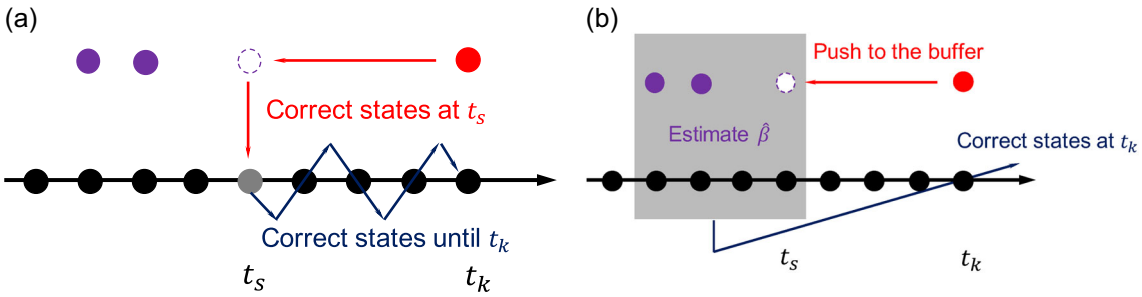

FIGURE 17 The sketches of EKF delay handler (EKF-DH) and visual model-predictive localization's (VML) handling delay mechanism. (a) The sketch of the EKF-DH proposed in Weiss et al. (2012). When the measurement arrives at t_k, EKF-DH first corrects the corresponding states at t_s and then updates the states until t_k. (b) The sketch of VML's mechanism of handling delay. When the measurement arrives, it will be pushed to the buffer with the corresponding states. Then, the error model will be estimated by the RANSAC approach. At last, the estimated model will be used to compensate the prediction at t_k. There is no need to update all the states between t_s and t_k [Color figure can be viewed at wileyonlinelibrary.com]

process, which can be caused by system interrupts. Thus, we first exclude the outliers by the Interquartile Range Method (Upton & Cook, 1996) and then provide the statistics for each component. The result can be found in Figure 22 and Table 3.

From Table 3, it can be seen that vision takes much more time than the other three parts. Please note though that the snake gate computer vision detection algorithm is already a very efficient gate detection algorithm. In fact, it has tunable parameters, that is, the number of samples

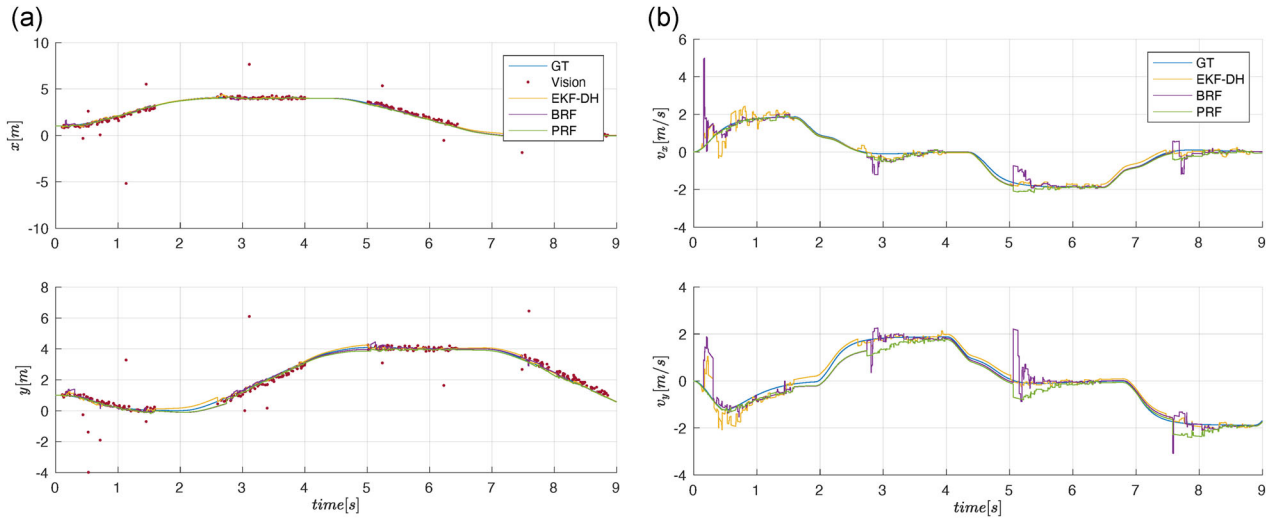

FIGURE 18 An example of the performance of the three filters when outliers and delay exist. (a) Position estimation of the three filters with outliers and delay. (b) Velocity estimation of the three filters with outliers and delay [Color figure can be viewed at wileyonlinelibrary.com]

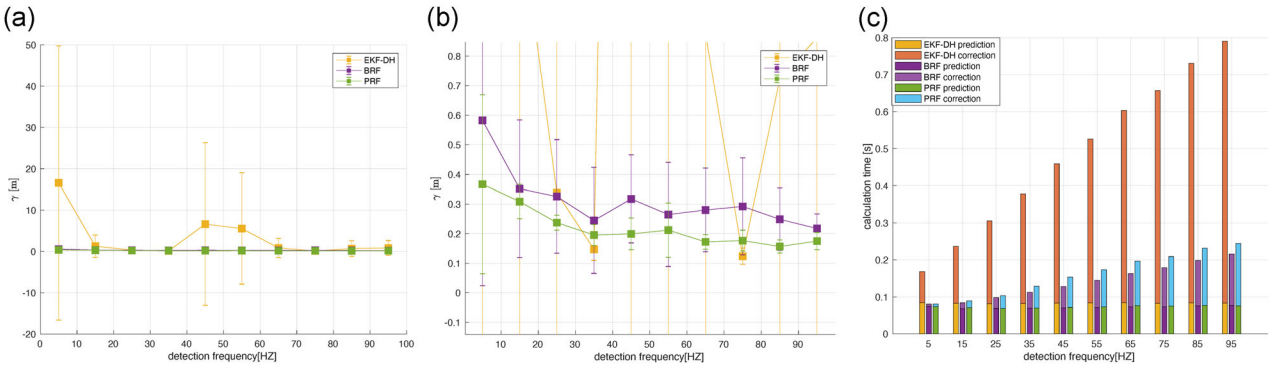

FIGURE 19 The estimation error of EKF delay handler (EKF-DH), Basic RANSAC Fitting (BRF), and Prior RANSAC fitting (PRF) and their calculation time with outliers and delay. (a) Estimation error of the EKF-DH, BRF, and PRF with different detection frequencies. (b) Partial enlarged drawing of (a). (c) Calculation time of the filters [Color figure can be viewed at wileyonlinelibrary.com]

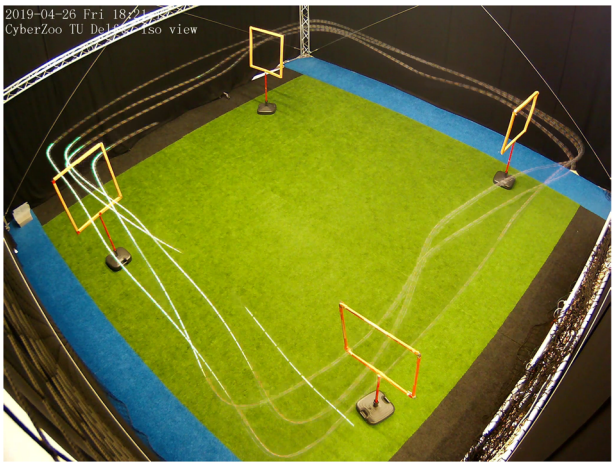

FIGURE 20 The picture of the Trashcan flying the track where the gates are displaced. The average speed is 2 m/s and the maximum speed is 2.6 m/s [Color figure can be viewed at wileyonlinelibrary.com]

taken per image for the detection (3,000 in the current setup), which allow the algorithm to run even much faster at the cost of having less accuracy (see S. Li et al., 2018 for more details). The main gain in time in the approach presented in this article is that we do not employ VIO and SLAM, which would take substantially more processing. However, as the Snake gate detection provides relatively low-frequency and noisy position measurements, the VML needs to run in high frequency and cope with the detection noise to still provide accurate estimation for the controller.

5.2 | Flying experiment without gate displacement

Figure 23 shows the flying result of the drone flying the track without gate displacement. The position of the 4 gates is listed in Table 4. In Table 4, x_g and y_g are the position of the gates in the real world and $\tilde{x}_g$ and $\tilde{y}_g$ are their position on the map. In this situation, they are the same. The aim of this experiment is to test the filter's performance with sufficient detections. Thus, the velocity is set to be 1.5 m/s to give the drone more time to detect the gate. In Figure 23, the blue curve is the ground truth data from OptiTrack motion capture system and the yellow curves are the filtering results. From the flying result, it can be seen that the filtered results are smooth and coincide with the ground truth position well. During the period when the detections are not available, the state prediction is still accurate enough to navigate the drone to the next gate. When the drone detects the next gate, the filter will correct the prediction. In this situation, the divergence of the states is only caused by the prediction drift. It should also be noted that when the outliers appears at 84 s, the filter is not affected by them because of the RANSAC technique in the filter.

5.3 | Flying experiment with gate displacement

In this section, we test our strategy under a difficult condition where the drone flies faster, the gates are displaced and the detection frequency is low. The real gate positions and their position on the map are listed in Table 5 and shown in Figure 24a. Gates are displaced between 0 and 1.5 m from their supposed positions. The dashed orange lines in Figure 24a denote the gate positions on the map while the solid orange lines denote the real gate positions which are displaced from the map. Figure 24b shows the flight data of the first lap. The orange solid gates are the ground truth positions of the gates. The yellow curve is the filtered position based on the gates' positions on the map (orange dashed gates). In other words, the yellow curve is where the drone thinks it is based on the knowledge of the map. After passing through one gate, when the drone detects

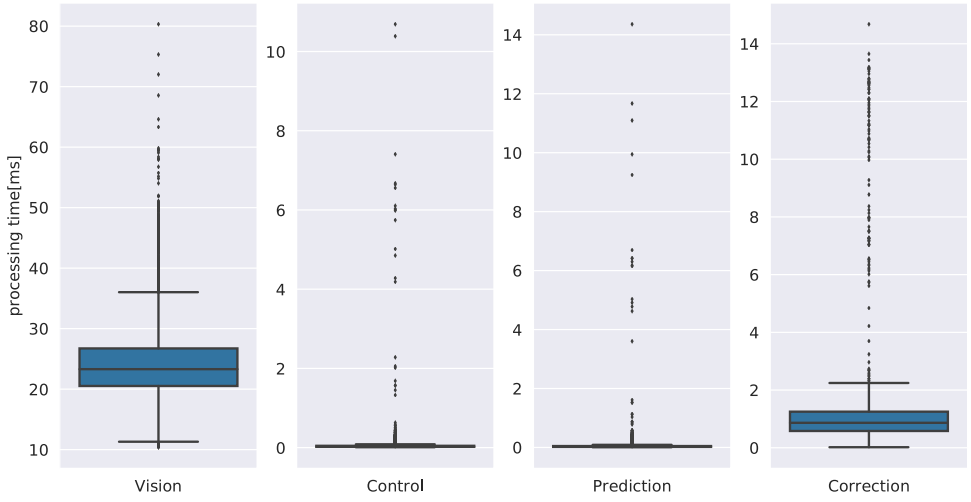

FIGURE 21 The statistical result of the four components' (vision, control, prediction and correction) processing time [Color figure can be viewed at wileyonlinelibrary.com]

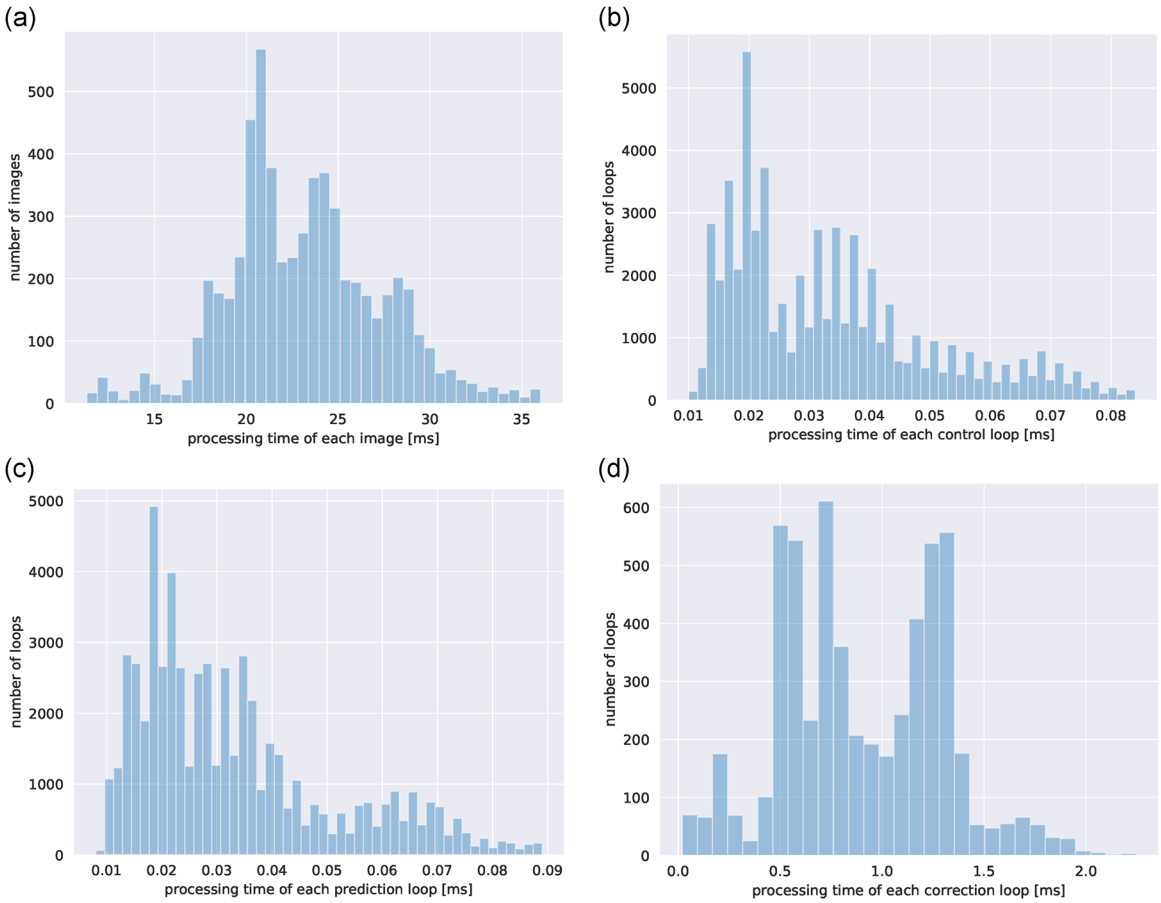

FIGURE 22 The histograms of each component's processing time without outliers. (a) Histogram of the vision's processing time. (b) Histogram of the controller's processing time. (c) Histogram of the visual model-predictive localization (VML)-prediction's processing time. (d) Histogram of the VML-correction's processing time [Color figure can be viewed at wileyonlinelibrary.com]

the next gate, the filter will start correcting the filtering error from the prediction error and the gate displacement.

The whole flight result is shown in Figure 25. From the result, it can be seen that the drone can fly the track for three laps with an average speed of 2 m/s and a maximum speed of 2.6 m/s while an experienced pilot flies the same drone in the same track with an average speed of 2.7 m/s after several runs of training. Figure 25a is the filtering result of the position. It should be noted that the filtering result does not coincide with the ground truth curve because of the displacement of the gates. The pose estimation is based on the gates' position on the map. When the gates are displaced, the drone still thinks they are at the position which the map indicates. After the turn, when the drone sees the next gate, which is displaced, it will attribute the misalignment to the prediction error and correct the prediction by means of new detections. With this strategy, our algorithm is robust to the displacement of the gates.

5.4 | Flying experiment with different altitude and moving gate

We also show a more challenging trace track where the height of the gates varies from 0.5 to 2.5 m. Also, during the flight, the position of the second gate (2.5 m) is changed after the drone passes through it. In the next lap, the drone can adapt to the changing position of the gate (Figure 26).

The flight result is shown in Figure 27. In this flight, the waypoints are not changed and the gates are deployed without any

	Mean (ms)	Std (ms)	Max. (ms)	Min. (ms)	Outlier rate (%)
Vision	23.2	4.2	36.0	11.3	6.7
Controller	0.03	0.02	0.08	0.01	6.4
VML-prediction	0.03	0.02	0.09	0.01	7.5
VML-correction	0.90	0.39	2.25	0.02	2.1

TABLE 3 Statistics for visual model-predictive localization's (VML) components' processing time without outliers

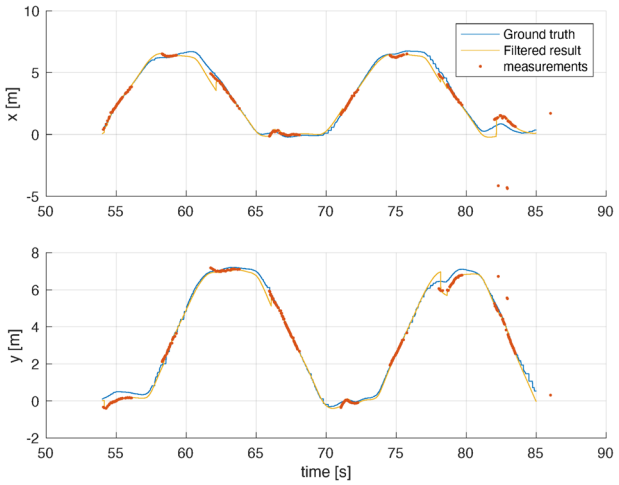

FIGURE 23 The flying result of the drone flying the track without the gate displacement [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 4 The position of the gates without displacement

Gate ID	x_g (m)	y_g (m)	$\tilde{x}_g$ (m)	$\tilde{y}_g$ (m)
1	5	0	5	0
2	6.5	5	6.5	5
3	1	7	1	7
4	0	1	0	1

TABLE 5 The position of the gates with displacement

Gate ID	x_g (m)	y_g (m)	$\tilde{x}_g$ (m)	$\tilde{y}_g$ (m)
1	5	0	4	0
2	6.5	5	5	5
3	1	7	1	6
4	0	1	0	1

ground truth measurement. Thus, the estimated position does not coincide with the ground truth position. It should be noted that the height difference between the second gate and the third gate is 2 m. With this altitude change which violates the constant altitude assumption for the prediction error model, the proposed VML is still accurate enough to navigate the drone through the gate.

From the real flight result, we can see that the VML performs well and can navigate the drone through the racing track with high speed even though the gates are displaced. Also, this strategy does not need computationally expensive methods like generic VIO and SLAM. This allows it to be run on a very light-weight flying platform.

6 | DISCUSSION

In this paper, we proposed a novel state estimation method called VML which provides navigation information for a 72-g autonomous racing drone. The algorithm's properties were thoroughly studied in simulation and the feasibility of real-world implementation was shown in challenging real-world experiments. Although in this paper VML is used for a specific drone race scenario, this method can be directly used for navigation in other more general scenarios where the sensors have low frequency, temporary failure, outliers and delays. For example, our approach can be directly adopted into an outdoor environment where position measurements are provided by a GPS signal that has a delay, temporary failures and outliers. Just as in our drone race experiments, the proposed approach should be more reliable than a Kalman filter. For indoor flight, we used a common linear drag model for state prediction which does not need a lot of effort and precise equipment to identify. Outdoor flight would require adaptations to this model, for instance such as the ones explained in, for example, Sikkel, de Croon, De Wagter, and Chu (2016).

We implemented our approach by adding a cheap smart camera Jevois to a tiny racing drone Trashcan. With very limited

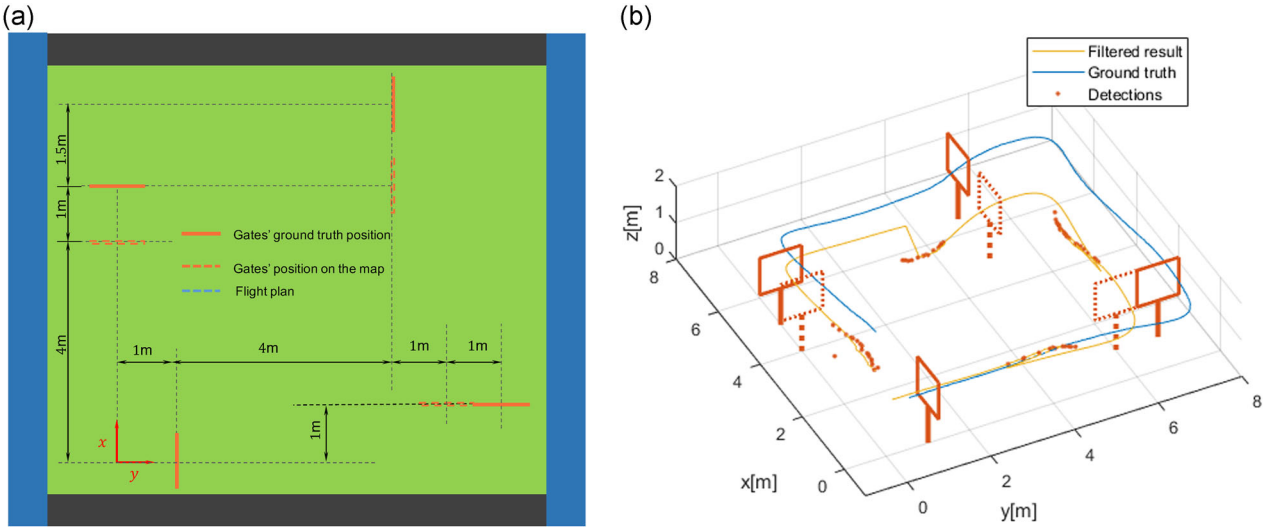

FIGURE 24 The experiment where the gates are displaced. When the drone sees the next gate after passing through one gate, the filter will start correcting the error caused by the prediction drift and the gate's displacement. Thus, there is a jump in the filtering result. (a) The map of the race track where the gates in the real world are displaced. (b) The flying data of the first lap [Color figure can be viewed at wileyonlinelibrary.com]

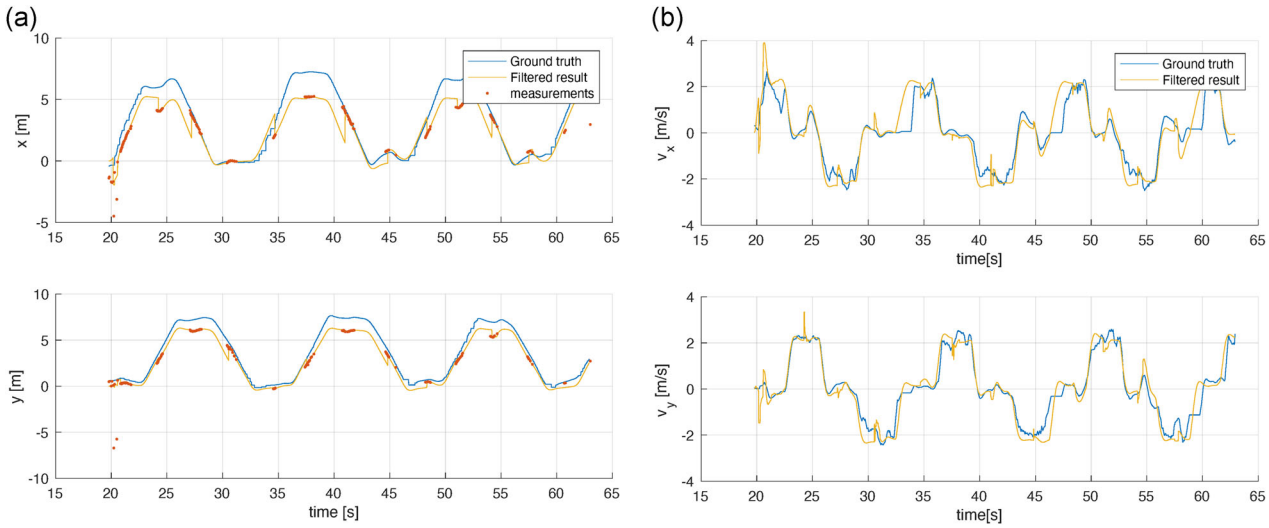

FIGURE 25 The result of flying the track with the gate displacement. (a) The position estimation result. It should be noted that the position estimation curve does not coincide with the ground truth curve coming from our motion capture system because the gate displacements. (b) The velocity estimation result of visual model-predictive localization [Color figure can be viewed at wileyonlinelibrary.com]

carrying capacity and more complex aerodynamics properties, it is still demonstrated that this light-weight flying platform has the ability to finish the drone race task autonomously. Compared with a regular size racing drone, the Trashcan has more complex aerodynamics and is more sensitive to disturbances. On the other hand, it has faster dynamics which can make maneuvers more agile. More important, it is much safer than a regular size racing drone, which may even allow for flying at home. In any case, the present approach represents another direction of the autonomous drone race, which does not need high performance and heavy onboard computers. Also, without computationally expensive navigation methods such as SLAM and VIO, the proposed approach is still able to make the drone navigate autonomously with relatively high speed.

However, the proposed approach still has its limitations. First of all, in this approach, we don't estimate the thrust. Instead, we use a non-changing altitude assumption to approximate the thrust to derive the prediction error model. The simulation and real-world experiments have shown that violating this assumption still results in accurate estimation. However, when the racing track will contain more considerable height changes, it may become desirable to estimate the thrust with a model, to have a more accurate error model and increase the estimation accuracy, especially in more aggressive flight.

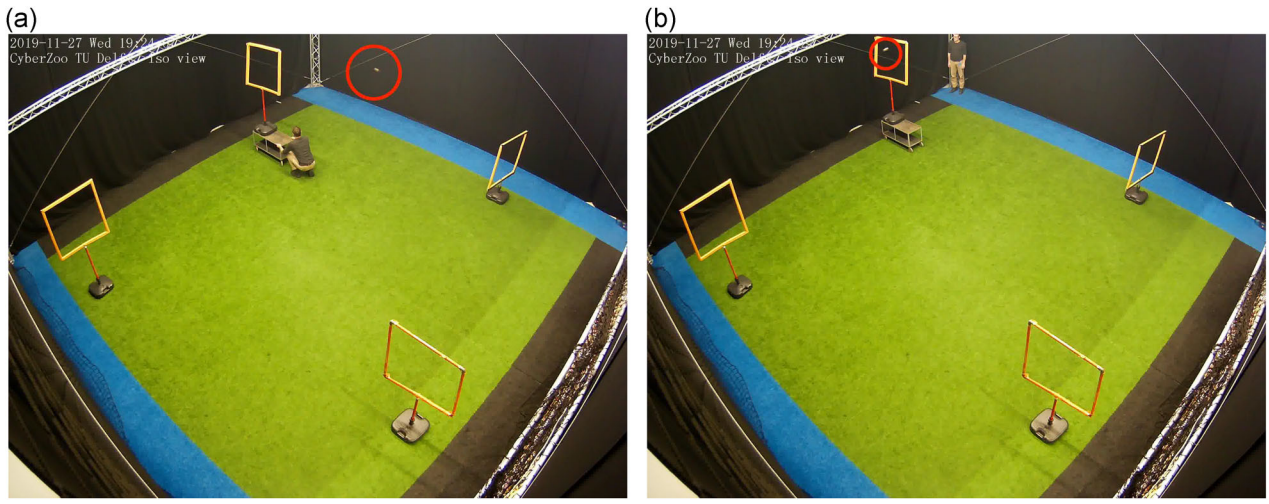

FIGURE 26 The flying experiment where the heights of the gates vary from 0.5 to 2.5 m. During the flight, the position of the second gate is changed. (a) After the drone passes through the second gate, the gate is moved. (b) In the next lap, the drone can adapt to the changing position of the gate and fly through it [Color figure can be viewed at wileyonlinelibrary.com]

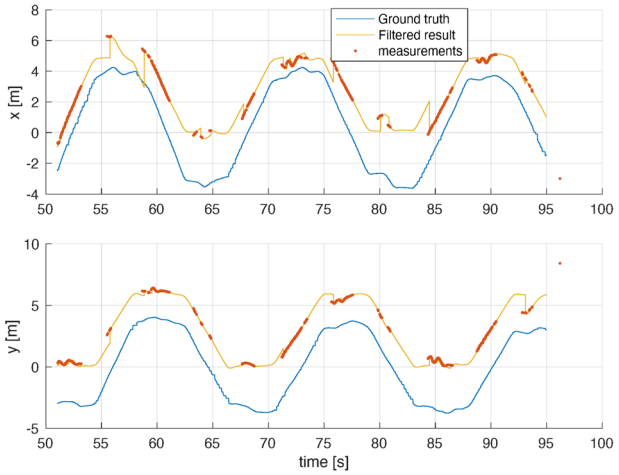

FIGURE 27 The flying result of the drone flying the track with different height and the gate's position changing [Color figure can be viewed at wileyonlinelibrary.com]

Second, the current detection method is sensitive to light conditions. Most failures are caused by the nondetection of the gate. This is a major bottleneck of increasing the speed of the flight. In the future, we will design a gate detection method using deep learning methods to detect the gate in a more complex environment. This deep net can then run on the GPU of the Jevois. Also, higher speeds could be attainable.

Thirdly, in this paper, we mainly focus on the navigation part of the drone. The guidance is only a waypoint based method and the controller is a PID controller. To make the drone fly faster, optimal guidance and control methods are needed (S. Li, Ozturk, De Wagter, de Croon, & Izzo, 2019; Tailor & Izzo, 2019; Tang, Sun, & Hauser, 2018). Another direction is to explore joint estimation for navigation. This will become very useful when one assumes that gates are mostly not displaced. Then, over multiple laps, the drone can get a better idea of where the gates are.

In the future, with the high speed development of computational capacity, when more reliable gate detection and online optimal control are implemented onboard, the speed of this autonomous racing drone can certainly be increased significantly. Compared with regularly sized drones, this tiny flying platform will be able to perform faster and more agile flight. At that time, the proposed VML approach will still be suitable for providing stable state estimation for the drone.

7 | CONCLUSION

In this paper, we presented an efficient VML approach to autonomous drone racing. The approach employs a velocity-stable model that predicts lateral accelerations based on attitude estimates from the AHRS. Vision is used for detecting gates in the image, and—by means of their supposed location in the map—for localizing the drone in the coarse global map. Simulation and real-world flight experiments show that VML can provide robust estimates with sparse visual measurements and large outliers. This robust and computationally very efficient approach was tested on an extremely lightweight flying platform, that is, a Trashcan racing drone with a Jevois camera. In the flight experiments, the Trashcan flew a track of three laps with an average speed of 2 m/s and a maximum speed of 2.6 m/s. To the best of our knowledge, it is the world's smallest autonomous racing drone with a weight six times lighter than the currently lightest autonomous racing drone setup, while its velocity is on a par with the currently fastest autonomously flying racing drones seen at the latest IROS autonomous drone race.

ORCID

Shuo Li http://orcid.org/0000-0002-8656-4661
Christophe De Wagter https://orcid.org/0000-0002-6795-8454
Guido C. H. E. de Croon http://orcid.org/0000-0001-8265-1496

REFERENCES

Chang, G. (2014). Robust Kalman filtering based on Mahalanobis distance as outlier judging criterion. *Journal of Geodesy*, *88*(4), 391–401.

Diderrich, G. T. (1985). The Kalman filter from the perspective of Goldberger' Theil estimators. *The American Statistician*, *39*(3), 193–198.

Faessler, M., Franchi, A., & Scaramuzza, D. (2017). Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories. *IEEE Robotics and Automation Letters*, *3*(2), 620–626.

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

Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, *24*(6), 381–395.

Gao, F., Wang, L., Wang, K., Wu, W., Zhou, B., Han, L., & Shen, S. (2019). Optimal trajectory generation for quadrotor teach-and-repeat. *IEEE Robotics and Automation Letters*, *4*(2), 1493–1500.

Gati, B. (2013). Open source autopilot for academic research—the paparazzi system. In *2013 American Control Conference*, IEEE, pp. 1478–1481.

Gross, J. N., Gu, Y., Rhudy, M. B., Gururajan, S., & Napolitano, M. R. (2012). Flight-test evaluation of sensor fusion algorithms for attitude estimation. *IEEE Transactions on Aerospace and Electronic Systems*, *48*(3), 2128–2139.

Hattenberger, G., Bronz, M., & Gorraz, M. (2014). Using the paparazzi UAV system for scientific research. In *IMAV 2014, International Micro Air Vehicle Conference and Competition 2014*, p. 247.

Jung, S., Cho, S., Lee, D., Lee, H., & Shim, D. H. (2018). A direct visual servoing-based framework for the 2016 IROS autonomous drone racing challenge. *Journal of Field Robotics*, *35*(1), 146–166.

Kaufmann, E., Gehrig, M., Foehn, P., Ranftl, R., Dosovitskiy, A., Koltun, V., & Scaramuzza, D. (2018). Beauty and the beast: Optimal methods meet learning for drone racing. *arXiv preprint*, arXiv:1810.06224.

Kaufmann, E., Loquercio, A., Ranftl, R., Dosovitskiy, A., Koltun, V., & Scaramuzza, D. (2018). Deep drone racing: Learning agile flight in dynamic environments. *arXiv preprint*, arXiv:1806.08548.

Li, S., Ozo, M., De Wagter, C., & de Croon, G. (2018). Autonomous drone race: A computationally efficient vision-based navigation and control strategy. *arXiv preprint*, arXiv:1809.05958.

Li, S., Ozturk, E., De Wagter, C., de Croon, G. C., & Izzo, D. (2019). *Aggressive online control of a quadrotor via deep network representations of optimality principles. arXiv preprint*, arXiv:1912.07067.

Li, Z., Chang, G., Gao, J., Wang, J., & Hernandez, A. (2016). Gps/uwb/mems-imu tightly coupled navigation with improved robust Kalman filter. *Advances in Space Research*, *58*(11), 2424–2434.

Loianno, G., Brunner, C., McGrath, G., & Kumar, V. (2017). Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and imu. *IEEE Robotics and Automation Letters*, *2*(2), 404–411.

Lupashin, S., Hehn, M., Mueller, M. W., Schoellig, A. P., Sherback, M., & D'Andrea, R. (2014). A platform for aerial robotics research and demonstration: The flying machine arena. *Mechatronics*, *24*(1), 41–54.

McGuire, K., De Croon, G., De Wagter, C., Tuyls, K., & Kappen, H. (2017). Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone. *IEEE Robotics and Automation Letters*, *2*(2), 1070–1076.

Mellinger, D., & Kumar, V. (2011). Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, IEEE, pp. 2520–2525.

Mellinger, D., Michael, N., & Kumar, V. (2012). Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*, *31*(5), 664–674.

Moon, H., Martinez-Carranza, J., Cieslewski, T., Faessler, M., Falanga, D., Simovic, A., & Kim, S. J. (2019). Challenges and implemented technologies used in autonomous drone racing. *Intelligent Service Robotics*, 1–12.

Moon, H., Sun, Y., Baltes, J., & Kim, S. J. (2017). The IROS 2016 competitions [competitions]. *IEEE Robotics & Automation Magazine*, *24*(1), 20–29.

Morrell, B., Rigter, M., Merewether, G., Reid, R., Thakker, R., Tzanetos, T., & Chamitoff, G. (2018). Differential flatness transformations for aggressive quadrotor flight. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 1–7.

Mueller, M. W., Hamer, M., & D'Andrea, R. (2015). Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadrocopter state estimation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 1730–1736.

Sanket, N. J., Singh, C. D., Ganguly, K., Fermüller, C., & Aloimonos, Y. (2018). Gapflyt: Active vision based minimalist structure-less gap detection for quadrotor flight. *IEEE Robotics and Automation Letters*, *3*(4), 2799–2806.

Santamaria-Navarro, A., Loianno, G., Solà, J., Kumar, V., & Andrade-Cetto, J. (2018). Autonomous navigation of micro aerial vehicles using high-rate and low-cost sensors. *Autonomous Robots*, *42*(6), 1263–1280.

Santana, L. V., Brandao, A. S., & Sarcinelli-Filho, M. (2015). Outdoor waypoint navigation with the ar. drone quadrotor. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, pp. 303–311.

Sikkel, L., de Croon, G., De Wagter, C., & Chu, Q. (2016). A novel online model-based wind estimation approach for quadrotor micro air vehicles using low cost MEMs IMUs. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp. 2141–2146.

Tailor, D., & Izzo, D. (2019). Learning the optimal state-feedback via supervised imitation learning. *Astrodynamics*, *3*(4), 361–374.

Tang, G., Sun, W., & Hauser, K. (2018). Learning trajectories for real-time optimal control of quadrotors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp. 3620–3625.

Upton, G., & Cook, I. (1996). *Understanding statistics*. Oxford: Oxford University Press.

van Horssen, E., van Hooijdonk, J., Antunes, D., & Heemels, W. (2019). Event-and deadline-driven control of a self-localizing robot with vision-induced delays.IEEE Transactions on Industrial Electronics.

Weiss, S., Achtelik, M. W., Chli, M., & Siegwart, R. (2012). Versatile distributed pose estimation and sensor self-calibration for an autonomous MAV. In *2012 IEEE International Conference on Robotics and Automation*, IEEE, pp. 31–38.

How to cite this article: Li S, van der Horst E, Duernay P, De Wagter C, de Croon GCHE. Visual model-predictive localization for computationally efficient autonomous racing of a 72-g drone. *J Field Robotics*. 2020;1–26.
https://doi.org/10.1002/rob.21956

APPENDIX A

A.1 Kalman filter's prediction model

$$\begin{cases} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \end{bmatrix} \\ \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \end{bmatrix} = \begin{bmatrix} \cos\psi^m & -\sin\psi^m \\ \sin\psi^m & \cos\psi^m \end{bmatrix} \left\{ \begin{bmatrix} -g\sin\theta \\ g\cos\phi \end{bmatrix} + \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \begin{bmatrix} \cos\psi^m & \sin\psi^m \\ -\sin\psi^m & \cos\psi^m \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} \right\} \\ \begin{bmatrix} \dot{B}_N \\ \dot{B}_E \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} \phi \\ \theta \end{bmatrix} = \begin{bmatrix} \phi^m \\ \theta^m \end{bmatrix} + \begin{bmatrix} \cos\psi^m & \sin\psi^m \\ -\sin\psi^m & \cos\psi^m \end{bmatrix} \begin{bmatrix} B_N \\ B_E \end{bmatrix} \end{cases}. \tag{A1}$$

The inputs of the system A1 is the AHRS reading $\mathbf{u} = [\phi^m, \theta^m, \psi^m]^\intercal$. The states of the Extended Kalman filter are $\mathbf{X} = [x, y, v_x, v_y, B_N, B_E]^\intercal$. With the standard Extended Kalman filter procedure list below, the states of the system can be estimated.

A.2 Pseudocodes

Algorithm 1: *gate_assignment*

Input: $\Delta\bar{x}_k, \Delta\bar{y}_k$

Output: $\bar{x}_k, \bar{y}_k$

for $i = 1; i <= gate\_numbers; i++$ **do**

  $\bar{x}_k^i = \cos\psi_g^i \Delta\bar{x}_k - \sin\psi_g^i \Delta\bar{y}_k + x_g^i$

  $\bar{y}_k^i = \sin\psi_g^i \Delta\bar{x}_k + \cos\psi_g^i \Delta\bar{y}_k + y_g^i$

  $\Delta d_k^i = (\bar{x}_k^i - \hat{x}_k)^2 + (\bar{y}_k^i - \hat{y}_k)^2$

end

$j = \mathrm{argmin}_i \, \Delta d_k^i$

$\bar{x}_k = \bar{x}_k^j$

$\bar{y}_k = \bar{y}_k^j$

Algorithm 2: Basic RANSAC Fitting

Input: $\Delta \mathbf{x}^p_{k-q,k}$, $\Delta \mathbf{t}$

Output: $\hat{\beta} = \left[\Delta x^p_{k-q} \Delta v^p_{k-q}\right]$

for $i = 1; i <= iterations; i++$ **do**
  $\mathbf{sample\_id} = random\_integers(k-q, k, n^s)$
  $\Delta \mathbf{t}^s = \Delta \mathbf{t}[\mathbf{sample\_id}]$
  $\Delta \mathbf{x}^s_{k-q,k} = \Delta \mathbf{x}^p_{k-q,k}[\mathbf{sample\_id}]$
  $[\Delta x^p_{k-q_i}, \Delta v^p_{k-q_i}] = linear\_regression(\Delta \mathbf{t}^s, \Delta \mathbf{x}^s_{k-q,k})$
  for $j = k-q; j < k; j++$ **do**
    $\epsilon_j = \left\| \Delta v^p_{k-q_i}(\Delta t_j - \Delta t_{k-q}) + \Delta x^p_{k-q_i} - \Delta x^p_j \right\|_2$
    if $\epsilon_j > \sigma_{th}$ **then**
      $\varepsilon_i += \sigma_{th}$
    end
    else
      $\varepsilon_i += \epsilon_j$
    end
  end
  if $\varepsilon_i < \varepsilon_{min}$ **then**
    $\varepsilon_{min} = \varepsilon$
    $\Delta x^p_{k-q} = \Delta x^p_{k-q_i}$
    $\Delta v^p_{k-q} = \Delta v^p_{k-q_i}$
  end
end

Algorithm 3: Visual Model-predictive Localization

while *true* **do**
  $t_i = current\_time$
  $x_i^p += v_{xi}^p T_s$
  $y_i^p += v_{y_i}^p T_s$
  $v_{xi}^p += (-g\tan\theta - cv_{xi}^p)T_s$
  $v_{y_i}^p += (g\tan\phi - cv_{y_i}^p)T_s$
  $t_{k-q} = clear\_old\_elements\_in\_queue()$
  if *flagNewPoseEstimation* **then**
    $queue.front++$
    $queue.\mathbf{time}[queue.front] = t_i$
    $queue.\mathbf{x}^p[queue.front] = x_i^p$
    $queue.\mathbf{y}^p[queue.front] = y_i^p$
    $queue.\bar{\mathbf{x}}[queue.front] = \bar{x}_i$
    $queue.\bar{\mathbf{y}}[queue.front] = \bar{y}_i$
    $queue.size++$
    if $queue.size > N_{fit}$ **then**
      $[\Delta x_{k-q}^p, \Delta v_{x_{k-q}}^p, \Delta y_{k-q}^p, \Delta v_{y_{k-q}}^p] = filter\_correct()$
    end
  end
  $\hat{x}_i = x_i^p - [\Delta x_{k-q}^p + (t_i - t_{k-q})\Delta v_{x_{k-q}}^p]$
  $\hat{y}_i = y_i^p - [\Delta y_{k-q}^p + (t_i - t_{k-q})\Delta v_{y_{k-q}}^p]$
  $\hat{v}_{xi} = v_{xi}^p - \Delta v_{x_{k-q}}^p$
  $\hat{v}_{yi} = v_{y_i}^p - \Delta v_{y_{k-q}}^p$
end

Algorithm 4: filter_correct

Output: $\Delta x_{k-q}^p, \Delta v_{x_{k-q}}^p, \Delta y_{k-q}^p, \Delta v_{y_{k-q}}^p$
for $i = 1; i <= queue.size; i++$ **do**
  $\Delta\mathbf{t}_i = queue.\mathbf{time}[i] - queue.\mathbf{time}[1]$
  $\Delta\mathbf{x}_i^p = queue.\mathbf{x}^p[i] - queue.\bar{\mathbf{x}}[i]$
  $\Delta\mathbf{y}_i^p = queue.\mathbf{y}^p[i] - queue.\bar{\mathbf{y}}[i]$
end
$[\Delta x_{k-q}^p, \Delta v_{x_{k-q}}^p] = linear\_regression(\Delta\mathbf{t}, \Delta\mathbf{x}^p)$
$[\Delta y_{k-q}^p, \Delta v_{y_{k-q}}^p] = linear\_regression(\Delta\mathbf{t}, \Delta\mathbf{y}^p)$

Algorithm 5: *flight_plan*

Output: x^r, y^r, z^r, ψ^r

if $(\hat{x}_k - \mathbf{wp}_x[waypoint\_id])^2 + (\hat{y}_k - \mathbf{wp}_y[waypoint\_id])^2 < D_{switch\_wp}$ **then**
  $waypoint\_id++$
end
if $(\hat{x}_k - \mathbf{wp}_x[waypoint\_id])^2 + (\hat{y}_k - \mathbf{wp}_y[waypoint\_id])^2 < D_{turn}$ **then**
  $\psi_{sp} = \mathbf{wp}_\psi[waypoint\_id + 1])$
  $r_{cmd} = k_r(\psi_{sp} - \psi^r)$
  $\psi^r += r_{cmd}$
end
$x^r = \mathbf{wp}_x[waypoint\_id])$
$y^r = \mathbf{wp}_y[waypoint\_id])$
$z^r = \mathbf{wp}_z[waypoint\_id])$
$\psi^r = \psi_{ref}$