Obstacle Avoidance using Event-based Visual Sensor and Time-To-Contact Processing

Fabien Colonnier, Luca Della Vedova, Rodney Swee Huat Teo, Garrick Orchard
Temasek Laboratories, National University of Singapore, Singapore
fabien.colonnier@nus.edu.sg

Abstract
Optic Flow is known to be useful in detecting obstacles and measuring Time-To-Contact, while event-based vision sensors have recently emerged as an efficient low-latency alternative to traditional frame-based vision sensors. This paper combines these two areas to present a visual collision avoidance system based on Optic Flow and Time-To-Contact computation using an event-based sensor. For demonstration, a quadrotor was fitted with the system and collision avoidance was tested. The quadrotor is shown successfully evading obstacles while flying at speeds up to 2.5 m s\(^{-1}\). The quadrotor performs an evasive manoeuver which can either be a turn away from the obstacle, or a complete stop (if no safe forward path is detected). An example of an obstacle detection shows that the maximal Time-To-Contact error is below 1.2 s. A video of the different experiments is provided as supplementary data.

1 Introduction
Recently, Unmanned Aerial Vehicles (UAVs) have been finding increasingly widespread application including photography, film [Cheng, 2015], visual inspection [Nikolic et al., 2013], art [Schoellig et al., 2014], documenting athletes [Dasgupta et al., 2018], home delivery [Hoareau et al., 2017], and as rescue operations [Michael et al., 2014]. The use of UAVs requires robust perception of the environment to avoid obstacles in these different scenarios, especially at high speeds.

For large rotorcraft [Scherer et al., 2008] and winged aerial vehicles [Bry et al., 2012], LIDAR can be used with great efficacy, although its cost still proves prohibitive in many applications. On the other hand, visual sensing provides a cheaper, passive, lower power alternative, but requires more complex algorithms to achieve similar robustness. As speed increases, the quality of the visual data typically decreases due to motion blur, thus degrading system performance. Nevertheless, frame-based vision sensors have been shown to perform well to generate a collision free trajectory [Shen et al., 2012]. Limitations on the computational load and accuracy of the mapping leads to still use LIDAR for fast flight [Mohra et al., ].

More recently, deep learning techniques have been used in navigation [Gandhi et al., 2017] and to prevent collision [Loquercio et al., 2018].

Optic Flow (OF) is known to be used by insects for navigation [Land and Nilsson, 2012], leading researchers to attempt to mimic insect behaviours in artificial systems. The centering-reflex of bees navigating through a corridor suggested by [Srinivasan et al., 1996] has been reproduced on a mobile robot using an OF based controller [Santos-Victor and Sandini, 1997]. Others later improved the corridor navigation with more elaborated algorithms. Conroy et al. used a Wide-Field integration algorithm, previously presented for a hovering task [Humbert et al., 2007], to provide a navigation signal from local OF measurements [Conroy et al., 2009]. Zingg et al. used a LKT algorithm [Shi and Tomasi, 1994] to compute OF at 20 Hz [Zingg et al., 2010] and an estimation of the speed to navigate safely. Roubieu et al. managed to perform navigation in different corridor configurations accounting from new biological findings [Seres et al., 2008], using minimalistic bio-inspired OF sensors [Roubieu et al., 2014]. Some applications adopted a downward facing camera to compute OF with elaborate sensor fusion [Bristeau et al., 2011], or in combination with sonar measurements to provide a speed estimation [Honegger et al., 2013]. The first flying robot to perform obstacle avoidance using to OF was using the I2A algorithm [Srinivasan, 1994] onboard a lightweight fixed-wing vehicle [Zufferey and Floreano, 2006].

Computing the Time-To-Contact from OF measurements has already been demonstrated [Camus, 1995] and used in obstacle avoidance on ground vehicles [Coombs et al., 1998; Song and Huang, 2001]. Others based their obstacle detection on the Locust LGMD
A new class of vision sensors based on biological principles has recently emerged [Posch et al., 2014]. Such sensors are called event-based vision sensors, sometimes also referred to as silicon retinas and Dynamic Vision Sensors (DVSs). These sensors consist of an array of pixels which detect and report when their log-illumination has changed by more than a given threshold. The thresholds, different for increases and decreases of the light intensity can be tuned to change the sensitivity of the sensor. Several example of sensors are already available, the DVS [Lichtsteiner et al., 2008], the Davis240 [Berner et al., 2013] and ATIS [Posch et al., 2010]. These sensors have already shown to be useful for different applications like Visual odometry [Zhu et al., 2017; Rebecq et al., 2017b], SLAM [Rebecq et al., 2017a] and can compute Optic Flow [Rueckauer and Delbruck, 2016]. Some robotic implementation of this novel visual sensor technology has been made, for localization [Vidal et al., 2018] or vertical landing [Pijnacker Hordijk et al., 2018].

This work, inspired by [Clady et al., 2014], demonstrates that a MAV (Micro Aerial Vehicle) equipped with an event-based sensor (see figure 1) can detect obstacles and either stop or avoid them if possible, thanks to the computation of the Time-To-Contact. Section 2 described the visual algorithm, whereas Section 3 details the robot hardware and its controller. The experimental results are presented in Section 4.

2 Description of the visual algorithm

The algorithm can be decomposed into the main steps below, which are described in more details in the following subsections:

- Preprocess events to remove noise
- Compute the Optic Flow (OF) from events
- Estimate the Focus-Of-Expansion (FOE) from OF measurements
- Estimate a Time-To-Contact (TTC) for each OF measurement
- Combine TTC measurements within each ROI using a Kalman Filter to obtain a single TTC per ROI
- Average the TTC separately for ROIs in the left, center, and right of the visual scene.

2.1 Preprocessing

The first step is to filter events to remove noise and limit the event rate. We use a Refractory filter and a Nearest Neighbour filter. The Refractory filter discards the events occurring at a given pixel after an event, whereas the Nearest Neighbour filter removes the isolated events in space during a given time interval (see [Czech and Orchard, 2016] for details).

2.2 Optic Flow Computation

Optic flow is computed on the filtered events, using the “plane-fitting” method presented in [Benosman et al., 2014] and improved in [Barranco et al., 2014; Rueckauer and Delbruck, 2016]. This technique allows to measure the displacement of an edge in a small window (here a 3 × 3 pixels), each time a new event is received. An obvious limitation of the method is that it cannot retrieve the image motion, i.e. movement of any 3D points into the image plane, when no intensity gradients are present in the image. Moreover, the true OF, defined as the image motion for a pixel having an edge in the FOV, is also not achievable with this method. Indeed, the result is also limited by the aperture problem. It means an edge movement cannot completely be defined in a small window if its ends are not visible in the aperture. Therefore, only the projection of the true OF to the perpendicular direction to the edge is measured, called the normal OF (see figure 2 for the relation between true and normal OF).
2.3 Estimation of the Focus Of Expansion

By definition, the Focus Of Expansion is, in a translational movement, where all the true OF are coming from, meaning no movement is seen at this particular point of the FOV. In another formulation, the FOE is the point where all the true OF vectors’ direction are crossing each other. Since only the normal OF is available, another method is needed to retrieve the FOE position.

The method used here was previously presented in [Aung et al., 2018], where a Matlab implementation is publicly available. The only difference is in the separation of the ON and OFF pathways, which improves the OF accuracy but increases the memory resources.

The parameters used for preprocessing and OF computation are shown in Table 1.

### 2.4 Estimating Time-To-Contact

The TTC is calculated by combining the FOE and OF measurements. Let \( \nabla_n \) be the normal OF measured at the point \( p(x,y) \). By normal, it applies to the edge in the FOV. Let \( \vec{U} \) and \( \vec{T} \), be the true OF for the same measurement and the tangential OF, respectively.

\[
\vec{U} = \nabla_n + \vec{T} \Leftrightarrow \begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{1}
\]

Figure 2 shows a schematic view of the vectors. Assuming that the position of the FOE is known, it is then possible to recompute the vector \( \vec{U} \), as by definition the vector is aligned to the FOE for a divergence OF pattern, meaning:

\[ \vec{U} = k \cdot \vec{OF} \Leftrightarrow \begin{bmatrix} u_x \\ u_y \end{bmatrix} = k \cdot \begin{bmatrix} x - x_f \\ y - y_f \end{bmatrix} \tag{2} \]

The solution of the system composed of the equations (1),(2) and (3), can be expressed as follows:

\[
\begin{align*}
\begin{cases}
u_x = \frac{v_x}{v_x^2 + v_y^2} \\ u_x = \frac{v_x}{v_x^2 + v_y^2} \cdot (x - x_f) \\
u_y = \frac{v_y}{v_x^2 + v_y^2} \\ u_y = \frac{v_y}{v_x^2 + v_y^2} \cdot (y - y_f)
\end{cases}
\end{align*} \tag{4}
\]

if \( v_x \cdot (x - x_f) + v_y \cdot (y - y_f) \neq 0 \), which is false in the case of \( p \) being at the FOE location, if \( \vec{V}_n = \vec{0} \), or if \( \frac{v_x}{v_y} = -\frac{y - y_f}{x - x_f} \) meaning that \( \vec{V}_n \) and \( \vec{OF} \) are in opposite direction which does not happen in the case of the FOE.

From the definition of the TTC from Camus [Camus, 1995]:

\[
TTC = \frac{Z_c}{\dot{Z}_c} = \frac{y - y_f}{\dot{y}} = \frac{x - x_f}{\dot{x}} 
\]

where \( Z_c \) is the depth of the obstacle and \( \dot{Z}_c \) is the speed in the obstacle direction.
From the definition, \( \dot{x} = u_x \) and \( \dot{y} = u_y \) in the case of translation only, we got the expression of \( \text{TTC} \) as follows:

\[
\text{TTC} = \frac{v_x \cdot (x - x_f) + v_y \cdot (y - y_f)}{v_x^2 + v_y^2}
\]

(6)

as long as the condition \( v_x^2 + v_y^2 \neq 0 \) is validated, which is true because otherwise the normal OF is null and would not be computed.

2.5 Kalman Filtering the Time-To-Contact for each ROI

Within each ROI a Kalman Filter (KF) is used to estimate the ROI’s TTC from the noisy individual TTC measurements.

TTC can be calculated as \( \text{TTC} = D/V \), from which it follows that \( V = D/\text{TTC} \), where \( V \) and \( D \) are the robot speed and the distance to the obstacle, respectively. Therefore, if the speed is considered constant, the evolution of the Time-To-Contact is linear and its derivative is constant. The implementation of a KF is then straightforward.

The states of the KF are the TTC and its derivative.

\[
X = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \text{TTC} \\ \dot{\text{TTC}} \end{bmatrix}
\]

(7)

The prediction state and error covariance is:

\[
\begin{align*}
\hat{X}_k^- &= A \cdot \hat{X}_{k-1} \\
P_k^- &= A \cdot P_{k-1} \cdot A^T + Q
\end{align*}
\]

(8)

where \( A = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \), and \( \Delta t \) is the time elapsed between two successive OF measurements. \( P \) is the covariance matrix and \( Q \) the processing noise.

The Kalman gain is computed as follows:

\[
K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}
\]

(9)

where \( H = \begin{bmatrix} 1 & 0 \end{bmatrix} \) and \( R \) the measurement noise.

As the computation of the inverse matrix is a time consuming operation, it is important to reduce it. Here, let

\[
P_k^- = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}, \text{ then } HP_k^- H^T = p_{11}. \text{ Therefore}
\]

\[
K_k = \frac{1}{p_{11} + R} \begin{bmatrix} p_{11} \\ p_{21} \end{bmatrix}
\]

(10)

Finally, the estimate and the error covariance are processed:

\[
\begin{align*}
\hat{X}_k &= \hat{X}_k^- + K_k \cdot (\tau_k - H \hat{X}_{k-1}) \\
P_k &= P_k^- - K_k H P_k^-
\end{align*}
\]

(11)

(12)

where \( \tau_k \) is the measurement. \( H \hat{X}_{k-1} \) can be reduced to \( \hat{x}_{k-1} \).

As the application is to detect obstacle, the visual scene can contain different objects. Therefore, the sensor frame should be divided into different ROI in order to apply Kalman filters for each. However, a tradeoff should be found between small ROIs where the better it discriminate the object in the scene but the less data are used, limiting the filtering, or bigger ROIs with more accurate output in case of large objects but less accurate according to the object edge position. In this case, the chosen ROI size is \( 30 \times 30 \) pixels.

2.6 Combining ROIs to obtain TTC for left, center, and right

An ROI size of \( 30 \times 30 \) pixels was chosen, resulting in 48 ROI TTC values for the \( 240 \times 180 \) resolution of the DAVIS240C sensor used. In our experiment we restrict the quadcopter flight to a horizontal plane (it cannot rise or fall), which only allows escape routes to the left or right. We therefore estimate the TTC separately for the left, center, and right of the image. We denote these estimates as \( \text{TTC}_{\text{left}}, \text{TTC}_{\text{mid}}, \text{and TTC}_{\text{right}} \) respectively (Fig. 4).

![Figure 4: Example of rendering of the TTC Kalman filter computed for each ROI (30 x 30 pixels) with the averages \( \text{TTC}_{\text{left}}, \text{TTC}_{\text{mid}}, \text{and TTC}_{\text{right}} \) computed over 3 x 6, 4 x 6 and 3 x 6 ROIs, respectively.](image)

3 Implementation

3.1 Hardware description

The robotic platform used is a Parrot Bebop 2. Only the attitude controller has been tuned, compared to the commercial version, to ensure stability with the additional payload. The event-based sensor is a Davis240 mounted on a home made PCB with a FPGA, a Zynq from Xilinx. The FPGA is in charge of filtering the events, allowing a reduced load of the WiFi connection to the ground station. The payload also includes a small battery as
power supply for the sensor. Figure 5 shows the quadrotor equipped with the sensor and sums up the Robot controller with the signals communicated between the quadrotor and the ground station, which are both done through WiFi.

### 3.2 Robot controller

#### Constant speed regulation

As it has been highlighted before, the visual algorithm makes the assumption of constant speed, as stated in the Kalman filter processing. Here, the Bebop 2 is already equipped with an ultrasonic sensor and a camera pointing downward measuring the altitude and the optic flow respectively. From these two measurements, it computes the ground speed, which are sent to the ground station. A simple PID controller was implemented to regulate the speed of the quadrotor. Figure 5 sums up the control strategy and the sensors used.

#### Compensating the pitch and roll movements

A second assumption is that the OF experienced by the visual sensor is only translational. As a quadrotor uses mainly body rotations to change its direction, it experiences pitch and roll movements during transition phase and smaller one during constant speed to reject perturbations. A mechanical solution could have been to place the sensor on a gimbal. However, it would have increased the payload weight and the integration complexity. As a proof of concept, it has been chosen to implement two software compensations.

One solution is to use an inhibitory system that discards the TTC measurements after the setpoint changed. This solution is used from the hovering phase to the forward movement at the beginning and during the evading manoeuvres.

Figure 6: State machine describing the robot control logic. All the parameters’ values are summarized in the table 1 and \( \text{diff}^* = \text{TTC}_{\text{left}} - \text{TTC}_{\text{right}} \)

A second solution, also implemented, is to apply a low pass filter on the FOE estimation (see equation (13)) and compare it to the instantaneous value. If the two values...
are too far apart, it means that the robot is in a transition phase and the TTC computation should not be done with the measurements collected during this time. As the FOE estimation relies only on OF direction, its value depends only on the translational speed direction and on rotation, therefore its stability is a good approximation of the robot steadiness.

\[
\begin{align*}
g &= e^{-F_{cut}(t-t_{old})} \\
FOE_{filt} &= g \cdot FOE_{filt} + (1 - g) \cdot FOE
\end{align*}
\]  
(13)

where \(F_{cut}\) is the cutoff frequency.

**Robot control logic**

The Robot control logic is a finite state machine which updates its state according to the visual input measurements \(TTC_{left}, TTC_{mid}\) and \(TTC_{right}\) computed previously (see section 2). Figure 6 describes the Robot control logic in details. It shows that the acceleration and the evade manoeuver are in open loop as only the time is a condition to change the state. It should also be noticed that all the filters are reset in order to discard data accumulated during the inhibition phases. In the Forward movement state, there are 3 transitions possible according to the averaged TTC measurements, either evade left or right or stop. The speed as well as the thresholds are manually tuned and summarized in table 1.

**4 Experimental results**

**4.1 Scenario 1: Obstacle detection**

A first experiment assessed the quality of the measurement and the capability of the robot to stop in front of an obstacle. The robot is placed in front of a patterned board and its forward speed is set to \(1m.s^{-1}\). This low speed has been chosen to allow sufficient flight time in our Vicon room.

Figure 7 depicts the results of this first scenario where the stop TTC setpoint has been fixed at 1.5s. It shows that the measurements are converging toward the real values and follows its evolution. The maximal absolute error is 1.2s once the KF have converged, 1.8s after the inhibition phase (or after 7.2s on the graph).

**4.2 Scenario 2: Obstacle avoidance and stop**

The second experimental scenario is presented in figure 8 and tested indoor. The objective of this second experiment is to demonstrate the capability of the robot to avoid obstacle and still be able to stop in front of another obstacle. Figure 9 displays the results obtained for this experiment. As the obstacle is initially in the middle of the FOV, the three TTC measurements give different values; the difference between left and right triggers the evading manoeuver. Then, the KF are reset and follow the same convergence pattern until the stop command is triggered.

Figure 7: a) Picture of the Vicon room with the patterned board. b) Forward speed in the robot frame, reference, measurement from the Bebop and ground truth by the Vicon system in blue, orange and yellow, respectively. c) Average TTC and ground truth, computed from the event-based visual sensor and from the Vicon position measurements, respectively.

Figure 8: The avoidance scenario: go straight toward the obstacle (1) and detect the obstacle, avoid the obstacle (2), go straight toward a bigger obstacle and stop (3).
This second scenario was repeated outdoor. The video given as supplementary data displays the recording for both. The outdoor test was initially carried out to provide ground truth through GPS measurements, as given by the Bebop. However, our sensor placed on top of the robot interfered with the GPS signal, which was therefore made unusable.

5 Conclusion

In this article, we display an Optic Flow based obstacle detection based on the measurements of an event-based visual sensor. It shows its ability to discriminate the relative distance to object and therefore avoid them by doing a sideway movement under high speed conditions. The third dimension of the flight was not taken into consideration. An improvement could be to extend this work to 3D flight.

Acknowledgements

The authors would like to thank Zhou Yu for the help with the Parrot Bebop ROS interface.

References


Appendix

Choice of the subsampling of $M_{\text{prob}}$

From a computational point of view, an optimal subsampling can be found. Let $N$ be the number of times the probability matrix is updated for each event. The function is defined as $N(x) = \frac{mn}{x^2} + x^2$, where $x$ represents the width of the ROI. Indeed, it is the number of ROI per frame, i.e. the ratio of the size of the frame by the size of the ROI, added with the size of the ROI.

The derivative is then $N'(x) = \frac{2mn}{x^3} + 2x$. Finding the size of the ROI which minimizes the number of $M_{\text{prob}}$ updates is therefore the value of $x$ which cancels $N'(x)$:

$$N'(x) = 0 \iff \frac{2mn}{x^3} = 2x \iff 2m \cdot n = x^4 \iff x = \sqrt[4]{2m \cdot n}$$ (14)

As the second derivative of $N''(x)$, for $x = \sqrt[4]{2m \cdot n}$ is equal to:

$$N''(x) = \frac{6mn}{x^4} + 2 \Rightarrow N''(\sqrt[4]{2m \cdot n})$$ (15)

As $m$ and $n$ are positive integers, $N''(x) > 0$, and as there are no other solution to the equation $N'(x) = 0$, $x = \sqrt[4]{2m \cdot n}$, this solution is a global minimum.

For example, with a Davis sensor with a $240 \times 180$ frame, the closest integer is 15. The number of updates for this $240 \times 180$ frame would be 43200 for each OF measurement, whereas with this subsampled version, it is reduced to 417, which is around a hundred times less.