Quadrotor UAV 3D path planning with optical-flow-based obstacle avoidance

Quadrotor UAV 3D Path Planning with Optical-Flow-based Obstacle Avoidance

Giancarlo Allasia[1], Alessandro Rizzo[1], Kimon Valavanis[2]

***Abstract*— A real-time waypoint-based 3D local path planning algorithm is proposed for obstacle avoidance using the optical flow obtained by a frontal monocular camera mounted on a quadrotor UAV. The algorithm accounts for vertical and horizontal obstacle avoidance, as well as for avoidance of frontally approaching obstacles. Implementation and testing are carried out in the ROS environment and the algorithm effectiveness is demonstrated via Gazebo simulations. Real-time algorithm performance is also assessed through software profiling and in terms of worst case execution time using the NVIDIA Jetson TX1 and RaspberryPi 4 for hardware-in-the-loop (HIL) tests.**

I. Introduction

Unmanned aerial vehicles (UAVs) in general, and quadrotors in particular, have been used in a wide spectrum of applications due to their versatility, flexibility, and their ability to fly at very low altitudes. However, a major issue to be overcome is real-time obstacle avoidance, hence the path planning strategy used to carry out a specific application should encompass the ability for the UAV to be aware of its surroundings. This ability is enabled by the inclusion of multiple sensors aboard. Sonars, radars, laser scanners (LIDAR), cameras or any combination of them are being used to deal with this issue [1]. LIDAR provides the most accurate data / information for environment mapping and for determining an obstacle-free path [2]. Even if LIDAR sensors in last few years became cheaper, compared to cameras they still have an economical disadvantage, not to mention their greater weight and power consumption, making cameras more advantageous particularly for small-scale UAVs. Information gathered from cameras, however, needs to be extracted in real-time by means of computer vision algorithms, already widely used in robotics for perception. Among the several available computer vision techniques, the biologically inspired *optical flow* algorithm has been investigated widely in literature as a solution to support UAVs' local path planning strategy and endowing UAVs with obstacle avoidance capability by means of just a monocular camera. The majority of optical flow-based path planning algorithms, unfortunately, suffer from some common limitations. Table I shows a classification of aforementioned limitation found in most recent related literature.

Therefore, this paper proposes an optical-flow-based local 3D path planning algorithm for obstacle avoidance, that exploits also third dimension enabling the avoidance of ground and floating obstacle and is able to avoid frontally approaching obstacle by considering the expansion of the 2D optical flow velocity vector field generated by the obstacles in the field of view (FOV) of the camera. The algorithm is real-time implementable on the quadrotor's on-board hardware, and a thorough analysis of the algorithm's worst case execution time (WCET) is carried out along with code profiling for performance evaluation in a HIL test using NVIDIA Jetson TX1 and RaspberryPi 4 boards. In the problem stated in this paper, the quadrotor is assumed to navigate based on a provided list of waypoints and be endowed, beyond standard sensors used for control purposes, only with a cheap frontal monocular camera, from which information from surrounding obstacles is extracted by exploiting optical flow computer vision algorithm. The algorithm is tested in a simulated environment implemented using Robotic Operating System (ROS) [3] in combination with Gazebo open source simulator [4] and OpenCV Python library for optical flow implementation, in particular Farnebäck's dense optical flow method [5], [6]. The effectiveness of the obstacle avoidance strategy will be tested in Gazebo in three scenarios, shown in Figure 1.

II. Proposed solution

A. Rationale of proposed solution

Assuming that the overall mission is expressed in terms of a list of waypoints to follow assigned by a global path planning algorithm, the strategy consists in computing in real time an intermediate waypoint to avoid the obstacle based on the optical flow vector field produced by the frames coming from the onboard camera, as is shown in Figure 2.

In Figure 3, an example of what the onboard camera sees and the respective brightness-coded optical flow field magnitude (the brighter the pixel, the faster the optical movement of that pixel among subsequent frames) with OSD superimposed is shown.

Closer objects to the moving observer (i.e. the UAV) appear to move faster in the FOV with respect to the farthest ones. Hence, areas in which the OF field is stronger are considered high collision risk areas, therefore the intermediate waypoint should be placed away from those areas. For instance, if obstacles on the right are at a smaller distance from us they seem to move faster in FOV, appearing as strong OF field on the right, hence a movement to the left is required for avoidance, so an intermediate waypoint will be appropriately placed on the left. The same holds for vertical

[1]G. Allasia and A. Rizzo are with the Department of Electronics and Telecommunications, Corresponding author: Alessandro Rizzo alessandro.rizzo@polito.it

[2]Kimon Valavanis is with the Ritchie School of Engineering and Computer Science, University of Denver, Denver, CO 80210 – USA kimon.valavanis@du.edu

TABLE I: Classification of developed monocular optical-flow based obstacle avoidance methods. Columns represent, respectively, the referenced work with its publication year (Ref.), the type of robot intended for algorithm application (Robot), the optical flow algorithm used (OF Method), the dimensionality of obstacle avoidance control commands produced (2D/3D), validation of achieved results (Result) and main limitations (Limitations) classified as follows: A - 2D Obstacle avoidance (left/right control commands); B - Offboard computation C - restricted applicability (e.g. specifically structured environment, intrinsically limited); D - real-time implementable (or not specified), E: does not deal with frontal obstacles.

Ref.	**Robot**	**OF Method**	**2D/ 3D**	**Results**	**Limitations**				
					A	B	C	D	E
[7]	Fixed-wing	Sparse Pyramidal OF	2D	SIM, EXP	✗				✗
[8]	Quadrotor	Sparse Pyramidal OF, Visual APF	2D	SIM	✗	n/a		n/a	✗
[9]	Quadrotor	Dense Farnebäck OF	-	EXP	n/a	✗			✗
[10]	Generic	Horn and Schunck OF	2D	SIM	✗	n/a	✗	✗	✗
[11]	Quadrotor	Horn and Schunck OF	2D	SIM	✗	n/a	✗	n/a	✗
[12]	Quadrotor	Horn and Schunck OF	3D	SIM, EXP					
[13]	Quadrotor	Lukas-Kanade OF	2D	SIM	✗		✗		✗
[14]	Quadrotor	Sparse Lukas-Kanade OF	1D	EXP	✗	✗		✗	✗
[15]	Quadrotor	Mixed dense and sparse OF	3D	EXP		✗			✗

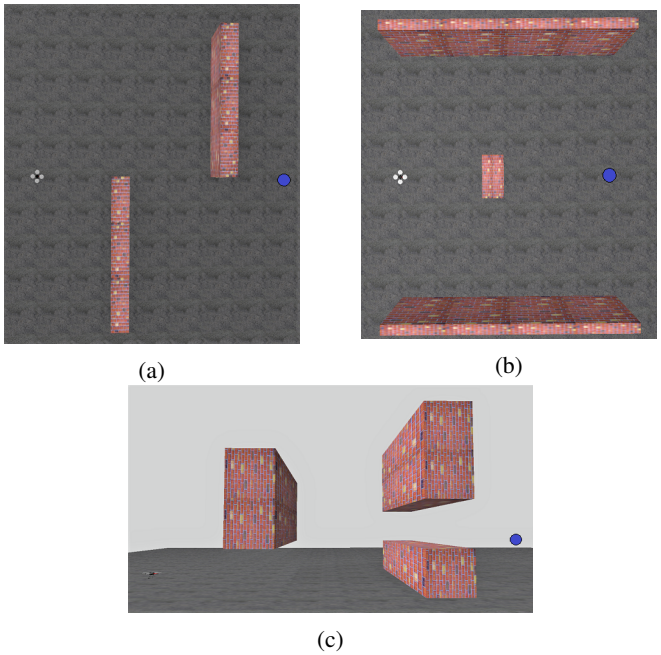


Figure 1: Test scenarios in Gazebo for horizontal avoidance (1a), vertical avoidance (1c) and frontal avoidance (1b). The blue circle represents the goal waypoint that the drone has to reach overcoming obstacles.

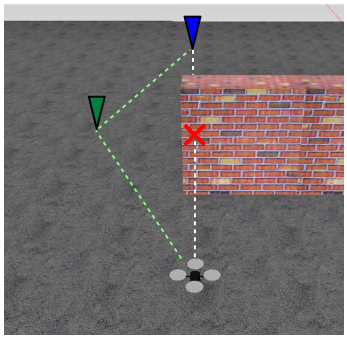
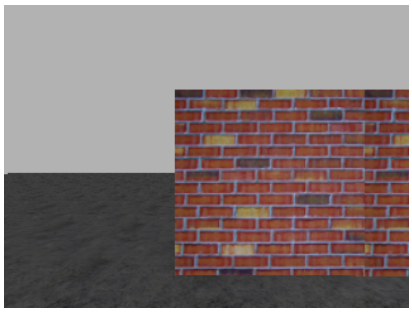
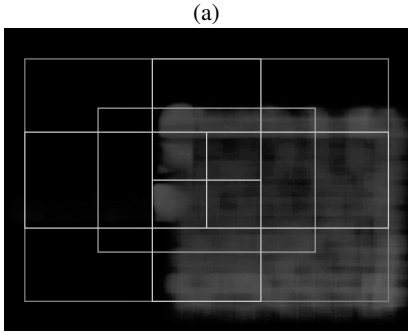


Figure 2: Here is shown the strategy's working principle. First the UAV is heading towards the next waypoint in the list (blue). When a lateral obstacle approaches the drone, from the right in this example, the obstacle's movement in drone's FOV leads to strong optical flow field on the right side of the FOV, triggering the obstacle avoidance, which computes an intermediate waypoint (green) that will become the next waypoint in the list, allowing the UAV to successfully avoid the obstacle.

obstacle avoidance. To translate this optical motion into a number, two scalar signals called horizontal and vertical optical flow unbalance signals are created, where sign and magnitude at a certain time instant indicate respectively where optical flow is more intense in the frame (left vs. right, up vs. down) and how strong is this unbalance. These signals are used to trigger an emergency situation whether the obstacle is approaching and avoid it by computing the intermediate waypoint. As far as frontally approaching



(a)



(b)

Figure 3: Figure 3a shows the onboard view of a lateral obstacle during forward motion is shown, while Figure 3b shows the computed OF in that scene with OSD superimposed, where brighter pixels correspond to faster moving points.

obstacles are concerned, given that these seem expanding in the FOV as they approach the observer, an appropriate signal called Expansion of Optical Flow (EOF) is defined, which is related to the divergence of the OF vector field. When one of these signals exceeds a threshold the obstacle avoidance strategy is triggered, with higher priority for frontal avoidance, and an intermediate waypoint is computed by using these signals to appropriately place it, by exploiting the spherical coordinates representation of the point to be computed in the vehicle reference frame. The produced waypoint is then prepended to the waypoint's list and the quadrotor heads to it. Once reached, the waypoint is deleted from the list and the quadrotor heads to the next one. The mathematical formalism of the algorithm is explained in the following sections with a bottom-up approach. After defining optical flow vector field in Section II-B, how horizontal and vertical optical flow unbalance and EOF signals are built is explained respectively in Section II-C and Section II-D. Then coordinates of the intermediate waypoint starting from aforementioned signals are derived in Section II-E. Software implementation is finally covered in Section II-G.

B. Optical flow

Optical flow in computer vision can be generally defined as the motion of objects in images between subsequent frames of image sequences. Such a motion is caused by the relative movement between the object and the camera expressed as a 2D velocity field, representing for each pixel its displacement between the two consecutive frames. Figure 4 shows how optical flow is defined.

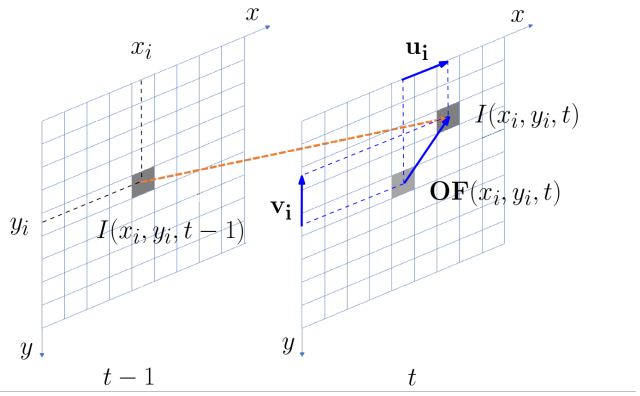


Figure 4: The displacement of a pixel obtained among subsequent frame defines the optical flow vector for that specific pixel. Image adapted from [12].

$I(x, y, t-1)$ and $I(x, y, t)$ are the gray scale versions of the two subsequent frames sampled from the camera, modeled as 2D matrix containing the brightness of pixels (x, y) respectively at time $t-1$ and t. These are matrices with dimensions $H \times W$, where $H = 240$ and $W = 320$ are respectively the height and width of the image. Given that the gray scale values are sampled over 8 bits, pixel brightness is an integer number ranging from 0 to 255. Time $t \in \mathbb{N}$ is discretized, with a sample rate of 10 Hz in the case of simulated onboard camera. Considering a single pixel (x_i, y_i), by solving an optimization problem that takes into account also the behaviour of that pixel's neighborhood among frames, it is possible to estimate its displacement from time $t-1$ to time t as a velocity vector $\mathbf{OF}(x_i, y_i, t) = u_i\hat{\mathbf{i}} + v_i\hat{\mathbf{j}}$. The solution to that optimization problem extended to all the pixels in the frame outputs a 2D velocity vector field, which represents the optical flow vector field and can be expressed as in Equation 1.

$$\mathbf{OF}(x, y, t) = u(x, y, t)\hat{\mathbf{i}} + v(x, y, t)\hat{\mathbf{j}} \tag{1}$$

Optical flow algorithms can be divided in sparse and dense, depending on whether flow is computed only for a subset of pixels (sparse), usually corresponding to feature detected by corner detecting methods or similar feature detection algorithms, or for all pixels in the image (dense). Sparse methods are computationally lighter but less accurate, while dense ones require more computation but manage to estimate the flow more correctly. In this paper the Gunnar Farnebäck dense optical flow method is used, implemented by OpenCV Python library. Farnebäck's method is not only more accurate than classic Lukas-Kanade method, as proved by results in [6] from a test on Yosemite sequence, but it gets faster on smaller images as tested in [16]. This last feature becomes important given that this algorithm applies the pyramid method, i.e. the optical flow computation is

applied at different levels of frames' downsizing to capture larger optical movement, improving robustness.

C. Optical flow unbalance computation

In order to obtain a measure of which areas in the FOV experience a stronger optical flow, the frames sampled from the onboard camera are divided into different templates, as can be seen in Figure 5, of which dimensions are determined by a trial and error procedure.

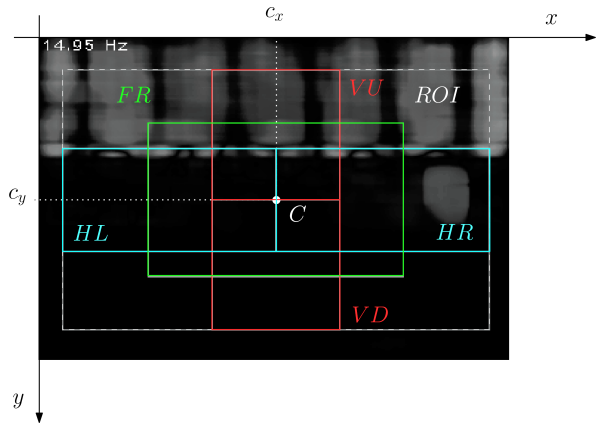


Figure 5: Templates geometry notation.

Not the whole frame area may be considered important for triggering collision avoidance. For instance, optical flow contributions from corners can be considered negligible for that purpose, as well as the most external strip, because, based on the assumption that that the camera is always facing forward motion direction, objects in those areas won't likely enter in collision trajectory. This is why a region of interest (ROI) smaller than frame's dimensions is defined, and templates are defined inside ROI. Templates are designed to cover a vertical strip (VU and VD) and an horizontal strip (HL and HR) for vertical and horizontal unbalance computation and obstacle avoidance, respectively. The same reasoning holds for the central template (FR) used for EOF computation. Vertical and horizontal optical flow unbalance signals are defined as:

$$e_V(t) = \sigma_{VD}(t) - \sigma_{VU}(t) \tag{2a}$$

$$e_H(t) = \sigma_{HR}(t) - \sigma_{HL}(t) \tag{2b}$$

where the signal $\sigma_T(t)$ associated to generic template T is the sum of compensated OF field magnitude for all pixels in T, defined as:

$$\sigma_T(t) = \sum_{(x,y)\in T} \|\mathbf{OF_C}(x,y,t)\| \tag{3}$$

$\mathbf{OF_C}$ is the field $\mathbf{OF}$ computed from the camera with adequate compensation for self-motion produced OF (more about this in Section II-F). In order to smooth the noisy unbalance signals and avoiding peaks that could trigger avoidance unnecessarily, a Moving Mean Filter (MMF) is applied. Filtered versions of unbalance signals are produced:

$$\overline{e_V}(t) = \frac{1}{M}\sum_{i=0}^{M-1} e_V(t-i), \quad M = 3 \tag{4a}$$

$$\overline{e_H}(t) = \frac{1}{M}\sum_{i=0}^{M-1} e_H(t-i), \quad M = 3 \tag{4b}$$

Length of filter's memory M has been set by trial and error procedure.

D. Expansion of Optical Flow (EOF) computation

When an object is approaching frontally, it seems for perspective reasons expanding in the FOV. In order to quantify this expansion, the concept of Expansion of Optical Flow is introduced. First, the Focus Of Expansion (FOE), that is the fixed point of the expansion, is considered. It is the point through which (theoretically) all directions of the vectors in a purely expanding field pass. Usually is not the case and its coordinates are obtained by solving an LSE problem, that leads to unreliable estimation. In this paper, FOE is assumed fixed in frame's center. Given that, ideally, FOE indicates the direction in which the observer is moving and in the presented work the drone always moves forward straight toward next waypoint, this assumption is reasonable. Then, for each pixel in the frontal template, the divergent component of OF from FOE is computed and divided by its distance from FOE to give more importance to points directly in front of the quadrotor, as shown in following equations and in Figure 6. Finally, all contributions from template's pixels are summed up to obtain the EOF signal.

$$OF_{DIV,i}(t) = \mathbf{OF_i}(t)\cdot\hat{\mathbf{u}}_{\mathbf{r,i}}, \quad \hat{\mathbf{u}}_{\mathbf{r,i}} = \frac{\mathbf{r_i}}{\|\mathbf{r_i}\|} \tag{5a}$$

$$EOF_i(t) = \frac{OF_{DIV,i}(t)}{\|\mathbf{r_i}\|} = \frac{\mathbf{OF_i}\cdot\mathbf{r_i}}{\|\mathbf{r_i}\|^2} \tag{5b}$$

$$EOF(t) = \sum_i^N EOF_i(t) \tag{5c}$$

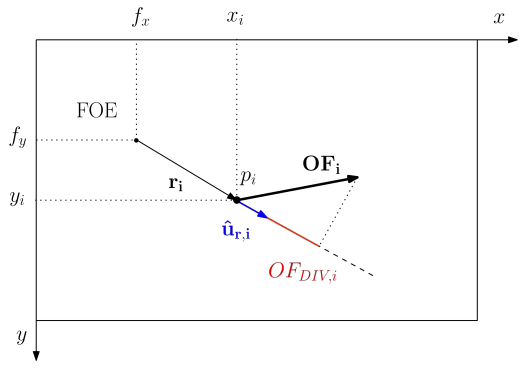


Figure 6: Computation of OF divergence component $OF_{DIV,i}$ for the i^{th} pixel.

E. Intermediate waypoint computation

By means of the obtained signals, the situation can be evaluated and action can be taken. Thresholds are defined and if any of the three signals exceeds them the obstacle avoidance strategy is triggered and the adequate intermediate waypoint is computed. This is done by using the signals to compute the spherical coordinates in the mobile (vehicle) reference frame and then translating it in cartesian coordinate in the fixed (world) reference frame, as shown in Figure 7.

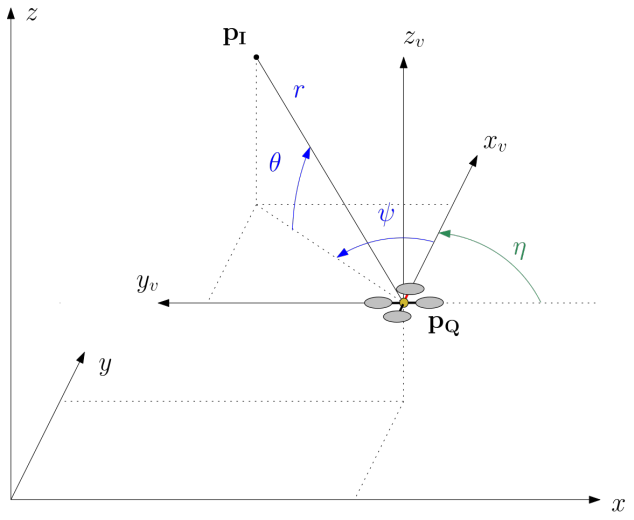


Figure 7: Intermediate waypoint computation for obstacle avoidance.

Vertical and horizontal unbalance signals and EOF signal are compared to respective thresholds τ_V, τ_H, τ_F to trigger the obstacle avoidance strategy. Depending on which kind if avoidance is required, horizontal/vertical or frontal, different equations for computing spherical coordinates in vehicle reference frame are used. The frontal avoidance has priority over horizontal and vertical avoidance. For instance, if both $EOF(t) > \tau_F$ and $\overline{e_V}(t) > \tau_V$ at a certain time instant t, only the frontal avoidance is applied and the second set of equations will be used. As far as horizontal or vertical avoidance is concerned, when $EOF(t) \leq \tau_F$, the Equations 6 for computing r, θ and ψ at time instant t are applied:

$$r = r_{V,H} \tag{6a}$$

$$\theta(t) = \begin{cases} K_{P,V}\ \overline{e_V}(t), & \text{if } |\overline{e_V}(t)| > \tau_V \\ 0, & \text{if } |\overline{e_V}(t)| \leq \tau_V \end{cases} \tag{6b}$$

$$\psi(t) = \begin{cases} K_{P,H}\ \overline{e_H}(t), & \text{if } |\overline{e_H}(t)| > \tau_H \\ 0, & \text{if } |\overline{e_H}(t)| \leq \tau_H \end{cases} \tag{6c}$$

Angles are then saturated to $\pm\frac{\pi}{2}$ in case the values exceed the range. For frontal avoidance instead, i.e. $EOF(t) > \tau_F$, a right angle turn to be avoided is required. Equations 7 are used for computing intermediate waypoint's spherical coordinates.

$$r = r_F \tag{7a}$$

$$\theta(t) = 0 \tag{7b}$$

$$\psi(t) = \begin{cases} -\frac{\pi}{2}, & \overline{e_H}(t) \leq 0 \\ +\frac{\pi}{2}, & \overline{e_H}(t) > 0 \end{cases} \tag{7c}$$

Finally, the translation in cartesian coordinates in the world reference frame is needed in order to feed this point as a reference in position to the quadrotor's controller. Defining $\eta(t)$ as the heading of the quadrotor with respect to world reference frame's x axis, $\mathbf{p_I}$ as the intermediate waypoint in world reference frame and $\mathbf{p_Q}$ as the quadrotor's position, coordinates of $\mathbf{p_I}$ are defined by Equations 8.

$$x_I(t) = r\cos\theta(t)\cos(\psi(t) + \eta(t)) + x_Q(t) \tag{8a}$$

$$y_I(t) = r\cos\theta(t)\sin(\psi(t) + \eta(t)) + y_Q(t) \tag{8b}$$

$$z_I(t) = r\sin\theta(t) + z_Q(t) \tag{8c}$$

The computed waypoint is now prepended to the stored waypoints list and will be used as the next reference in order to avoid the obstacle. The values for thresholds, gains and radii working for the simulation environment used in this work has been found by trial and error procedure.

F. Self-motion generated optical flow compensation

Given that the OF field generated is the result of the relative motion between the camera fixed to the UAV and the surrounding environment, quadrotor's tilting movements are influencing the generated optical motion. Given that only the OF generated by forward motion is useful to avoid obstacles, these unwanted OF must be compensated. To compute signals the compensated OF field $\mathbf{OF_C}$ is therefore taken into account, where compensation is operated by multiplying OF horizontal and vertical components for a time-varying coefficient as shown in Equations 9.

$$\mathbf{OF_C}(x,y,t) = u_C(x,y,t)\hat{\mathbf{i}} + v_C(x,y,t)\hat{\mathbf{j}} \tag{9a}$$

$$u_C(x,y,t) = u(x,y,t)\frac{1}{1 + K_{C,yaw}|\dot{\psi}(t)|} \tag{9b}$$

$$v_C(x,y,t) = v(x,y,t)\frac{1}{1 + K_{C,lin\ z}|\dot{h}(t)| + K_{C,pitch}|q(t)|} \tag{9c}$$

where $\dot{\psi}(t)$ is yaw rate at time t, $\dot{h}(t)$ is the climbing rate and $q(t)$ is the pitch rate. Coefficients' values have been tuned by means of a trial and error procedure as well.

G. Software implementation

To implement and test the algorithm, ROS/Gazebo quadrotor simulator package *hector_quadrotor* is used [17]. The algorithm is implemented as a node in ROS Melodic (Ubuntu 18.04 LTS) and set to run at a fixed rate of 5 Hz, even if higher rates can be achieved, as shown in Section III. In Figure 8, main topics used by the node are shown.

The algorithm at each iteration passes through three main phases: Status Evaluation, Control Command Computation,

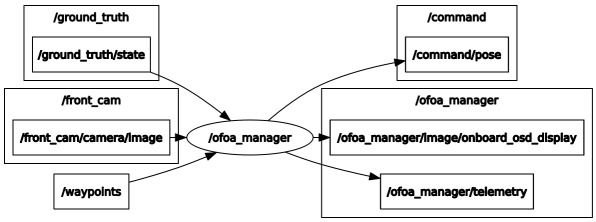


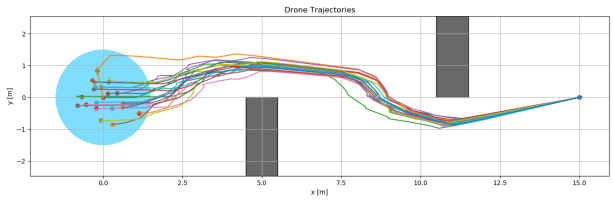
Figure 8: Graph of inputs and outputs for node */ofoa_manager* implementing optical-flow-based obstacle avoidance (OFOA) strategy.

Outputs Publication. In Status Evaluation OF field is computed as well as OF unbalances and EOF in order to compare signal to thresholds and set flags. In the Control Command Computation the previously evaluated flags are used to decide if and which kind of avoidance is required. In the last phase, Outputs Publication, produced outputs are published over respective topics.

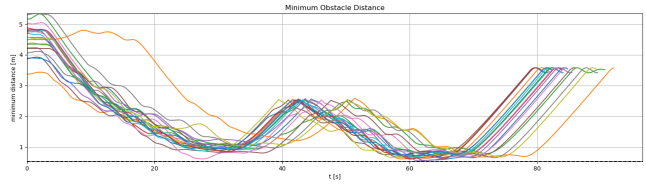
III. Results

A. Obstacle avoidance effectiveness in simulated test scenarios

For each scenario in Figure 1, 20 simulations are run with randomly generated starting points in the plane of interest according to a 2D Gaussian distribution; a further one with the origin $(0, 0, 0)$ as starting point is considered. Two kind of plots are produced. First one is a plot of all simulations' trajectories with obstacle profiles. An ellipse which principal semi-axes are 3σ of the randomly distributed starting point displacement along that direction is drawn. In the second plot, the minimum distance of the drone from the nearest obstacle for each run is shown. In order to compute success rate, the size of the quadrotor is considered. A certain threshold distance (dashed line) is set as the half of the size of the quadrotor along x or y in horizontal plane avoidance scenarios or along z in vertical plane avoidance scenarios, incremented of 30% as safety clearance. If the nearest obstacle is below aforementioned threshold, that run is labeled as a fail. Result plots for each scenario are shown in Figures 9, 10 and 11 and performance measurements in Table II. As can be seen from results, success rate is 100% for both horizontal and vertical avoidance and 95.2% for frontal avoidance.

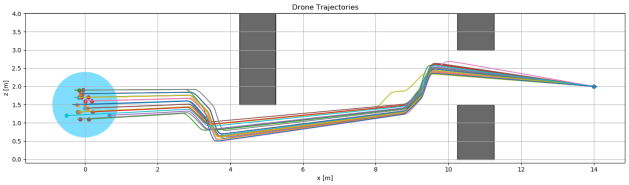


(a) Trajectories of all simulations.

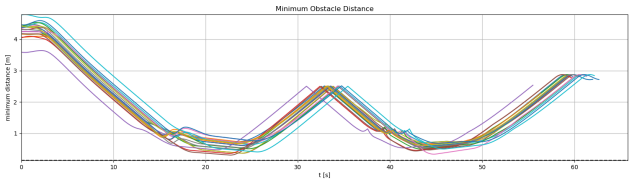


(b) Minimum distance form nearest obstacle.

Figure 9: Plots of results obtained from simulations run in lateral obstacle avoidance scenario in horizontal plane.



(a) Trajectories of all simulations.



(b) Minimum distance from nearest obstacle.

Figure 10: Plots of results obtained from simulations run in vertical obstacle avoidance scenario in vertical plane.

B. HIL simulations

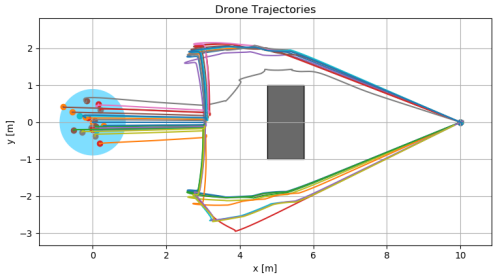
To prove that the algorithm can run in real-time on onboard integrable hardware, two boards are tested in a HIL setup, the NVIDIA Jetson TX1 and RaspberryPi 4. The experimental setup is shown in Figure 12 with Jetson TX1 as an example.

1) Worst Case Execution Time (WCET) analysis for real-time performances: To estimate WCET, a statistical analysis of algorithm's execution time is carried out on both platforms . In order to produce the measurements, a timer starts at the beginning of main algorithm function and it's stopped at the end of its execution, evaluating elapsed time and, by its inverse, the frequency of execution ideally achieved in that iteration. Here operating system time is considered, not simulation time. Measurements are collected over 60 seconds of simulation. The algorithm execution rate is fixed to $5\ Hz$, hence $200\ ms$ period, that is a safe enough value in order to let all the operations be executed without excessively compromising obstacle avoidance reactivity. Results are shown in Table III.

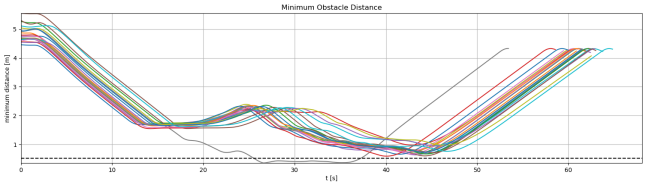
As shown, both boards are able to run the algorithm in real-time keeping up with the requested $5\ Hz$ rate. Jetson TX1 shows a WCET of $159.4\ ms$ (average $124.8\ ms$) corresponding to a theoretical minimum guaranteed rate of $6.27\ Hz$, while RaspberryPi 4 shows better performances with WCET of $78.4\ ms$ (average $66.8\ Hz$), corresponding to a minimum guaranteed rate of $12.75\ Hz$, being more than double the NVIDIA board's rate. Furthermore, RaspberryPi 4 weights around 52% of Jetson TX1's weight and its maximum consumption is around 41.7% in comparison. In

TABLE II: Summary of results from simulations run.

	Horizontal	**Vertical**	**Frontal**
Starting point	$X \sim \mathcal{N}(0, 0.5^2)$ m $Y \sim \mathcal{N}(0, 0.5^2)$ m	$X \sim \mathcal{N}(0, 0.3^2)$ m $Z \sim \mathcal{N}(1.5, 0.3^2)$ m	$X \sim \mathcal{N}(0, 0.3^2)$ m $Y \sim \mathcal{N}(0, 0.3^2)$ m
Minimum distance	0.54 m	0.31 m	0.59 m
Average minimum distance	0.66 m	0.53 m	0.73 m
Standard deviation	0.06 m	0.12 m	0.14 m
Success rate	**100 %**	**100 %**	**95.2 %**



(a) Trajectories of all simulations.



(b) Minimum distance form nearest obstacle.

Figure 11: Plots of results obtained from simulations run in frontal obstacle avoidance scenario in horizontal plane.

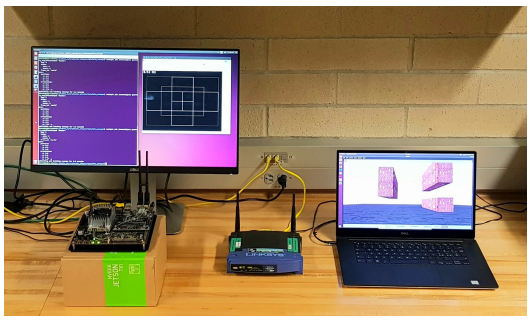


Figure 12: Setup for HIL simulation. On the left, the Jetson TX1 Development Kit is running Ubuntu 14.04 and OFOA ROS node. On the right, the laptop with virtualized Ubuntu 18.04 which hosts ROS master node and Gazebo simulator. The Jetson TX1 and the PC are connected by means of the WiFi router acting as a gateway, in the middle.

conclusion, the RaspberryPi 4 board not only can handle the real-time requirement better than the Jetson TX1 module, but it is more power efficient and lighter in weight, making it suitable to be mounted onboard on a quadrotor.

2) Code profiling: An analysis of where in the code the most part of execution time is spent to identify bottlenecks and try to optimize slower parts of code has to be carried out by means of Yappi profiler and results are visualized as

TABLE III: Statistics of main algorithm computation time performance run on NVIDIA Jetson TX1 and RaspberryPi 4 embedded systems. The equivalent rate is obtained as the inverse of execution time.

	Jetson TX1		RaspberryPi 4	
	Time [ms]	Rate [Hz]	Time [ms]	Rate [Hz]
Minimum	97.7	6.27	58.7	12.75
Maximum	159.4	10.24	78.4	17.02
Average	124.8	8.07	66.8	15.00
Standard deviation	11.9	0.78	2.9	0.63

treemaps in KCacheGrind. Experiment is carried out as HIL simulation with NVIDIA Jetson TX1 and results are shown in Figure 13.

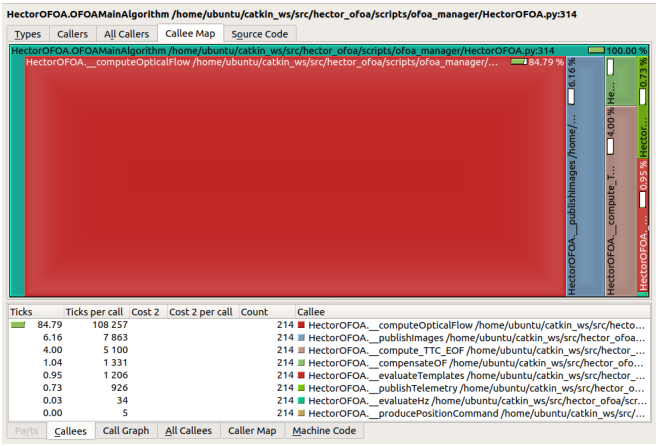


Figure 13: Treemap profiling representation by means of KCacheGrind of algorithm run on NVIDIA Jetson TX1.

What can be noticed is that the majority of time is spent in OF field computation, actually 84.79% of the overall time spent in caller function *OFOAMainAlgorithm*, making it the real bottleneck of the whole algorithm. OF field computation is implemented by an OpenCV function that recalls an optimized compiled function in C++, therefore cannot be optimized anymore. The second most expensive function with a time cost of 6.16% of main function execution time, hence already negligible, is the one used to publish outputs by means of standard non optimizable ROS Python library's functions. The remaining functions have a negligible impact and contain already optimized or non-optimizable operations.

IV. Conclusions

Effectiveness of proposed real-time waypoint-based 3D local path planning strategy in avoiding lateral, floating, ground and frontal obstacles has been demonstrated with high success rate. Capability of running in real-time at a rate of at least $5\ Hz$ on onboard integrable embedded hardware has been shown. The proposed algorithm, even if proved effective, has some limitations such as dependence on visibility (e.g. dark environment, poorly textured objects, limited FOV angle not perceiving obstacles on the sides such as wall corners) and lack of depth information due to the use of a monocular camera.

Acknowledgment

This work is partially supported by Amazon Science through a 2019 Amazon Research Award granted to Dr. A. Rizzo.

References

[1] J. N. Yasin, S. A. S. Mohamed, M. Haghbayan, J. Heikkonen, H. Tenhunen, and J. Plosila, "Unmanned aerial vehicles (uavs): Collision avoidance systems and approaches," *IEEE Access*, vol. 8, pp. 105 139–105 155, 2020.

[2] Lidar wikipedia page. [Online]. Available: https://en.wikipedia.org/wiki/Lidar

[3] Robotic operating system (ros). [Online]. Available: https://www.ros.org/

[4] Gazebo simulator tool. [Online]. Available: http://gazebosim.org/

[5] Opencv: an open source computer vision library. [Online]. Available: https://opencv.org/

[6] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," vol. 2749, 06 2003, pp. 363–370.

[7] B. Richards, S. Bhandari, M. Gan, J. Dayton, M. Enriquez, J. Liu, and J. Quintana, *Obstacle Avoidance System for UAVs using Computer Vision*, 2015. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2015-0986

[8] H. Miao and Y. Wang, "Optical flow based obstacle avoidance and path planning for quadrotor flight," 06 2018, pp. 631–638.

[9] P. Gao, D. Zhang, Q. Fang, and S. Jin, "Obstacle avoidance for micro quadrotor based on optical flow," in *2017 29th Chinese Control And Decision Conference (CCDC)*, 2017, pp. 4033–4037.

[10] P. Agrawal, A. Ratnoo, and D. Ghose, "A composite guidance strategy for optical flow based uav navigation," *IFAC Proceedings Volumes*, vol. 47, no. 1, pp. 1099 – 1103, 2014, 3rd International Conference on Advances in Control and Optimization of Dynamical Systems (2014). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667016327914

[11] A. Eresen, N. imamoğlu, and M. Efe, "Autonomous quadrotor flight with vision-based obstacle avoidance in virtual environment," *Expert Systems with Applications*, vol. 39, pp. 894–905, 01 2012.

[12] G. Cho, J. Kim, and H. Oh, "Vision-based obstacle avoidance strategies for mavs using optical flows in 3-d textured environments," *Sensors(Basel)*, 6 2019.

[13] D.-W. Yoo, D.-Y. Won, and M.-J. Tahk, "Optical flow based collision avoidance of multi-rotor uavs in urban environments," 2011.

[14] W. Aguilar, L. Álvarez, S. Grijalva, and I. Rojas, *Monocular Vision-Based Dynamic Moving Obstacles Detection and Avoidance*, 08 2019, pp. 386–398.

[15] C. Wang, W. Liu, and M. Q. . Meng, "Obstacle avoidance for quadrotor using improved method based on optical flow," in *2015 IEEE International Conference on Information and Automation*, 2015, pp. 1674–1679.

[16] J. de Boer and M. Kalksma, "Choosing between optical flow algorithms for uav position change measurement," 2015.

[17] Hector quadrotor ros wiki page. [Online]. Available: http://wiki.ros.org/hector$_quadrotor$