*remote sensing*

MDPI

*Article*

# Collision Avoidance on Unmanned Aerial Vehicles Using Neural Network Pipelines and Flow Clustering Techniques

**Dário Pedro [1,2,3,*], João P. Matos-Carvalho [4] and José M. Fonseca [2,3] and André Mora [2,3]**

[1] Projecto Desenvolvimento Manutenção Formação e Consultadoria, 1300-609 Lisbon, Portugal
[2] Center of Technology and Systems, UNINOVA, 2829-516 Caparica, Portugal; jmf@uninova.pt (J.M.F.); atm@uninova.pt (A.M.)
[3] Electrical Engineering Department, FCT, NOVA University of Lisbon, 2829-516 Caparica, Portugal
[4] Cognitive and People-Centric Computing Labs (COPELABS), Universidade Lusófona de Humanidades e Tecnologias, Campo Grande 376, 1749-024 Lisboa, Portugal; joao.matos.carvalho@ulusofona.pt
* Correspondence: dario.pedro@pdmfc.com

**Abstract:** Unmanned Autonomous Vehicles (UAV), while not a recent invention, have recently acquired a prominent position in many industries, and they are increasingly used not only by avid customers, but also in high-demand technical use-cases, and will have a significant societal effect in the coming years. However, the use of UAVs is fraught with significant safety threats, such as collisions with dynamic obstacles (other UAVs, birds, or randomly thrown objects). This research focuses on a safety problem that is often overlooked due to a lack of technology and solutions to address it: collisions with non-stationary objects. A novel approach is described that employs deep learning techniques to solve the computationally intensive problem of real-time collision avoidance with dynamic objects using off-the-shelf commercial vision sensors. The suggested approach's viability was corroborated by multiple experiments, firstly in simulation, and afterward in a concrete real-world case, that consists of dodging a thrown ball. A novel video dataset was created and made available for this purpose, and transfer learning was also tested, with positive results.

**Keywords:** UAVs; collision avoidance; clustering; artificial intelligence; machine learning; neural network; deep learning; collision prevention; drones



**Citation:** Pedro, D.; Matos-Carvalho, J.P.; Fonseca, J.M.; Mora, A. Collision Avoidance on Unmanned Aerial Vehicles Using Deep Neural Networks and Clustering Techniques with RGB Cameras. *Remote Sens.* **2021**, *13*, 2643. https://doi.org/10.3390/rs13132643

Academic Editor: Biswajeet Pradhan

Received: 19 May 2021
Accepted: 29 June 2021
Published: 5 July 2021



## 1. Introduction

UAVs evolved from an emerging innovation, to a common technology in the consumer and commercial sectors of the economy [1–3]. This paper addresses a safety key issue that, most of the time, is discarded by UAV manufacturers, but that is critical for a massive deploy in urban environments, to allow UAVs to achieve the same levels of autonomy as cars [4], which is the collision avoidance with highly dynamic objects. The proposed solution works in the symbiosis with standard autonomous fly and avoid architecture [5]. This level of safety and reliability must be maintained regardless of operating conditions or the occurrence of unanticipated events. Due to carelessness, and disregard for these type of events, multiple disasters have happen in the past [6–12], which will increase considerably with the expected exponential increase in the number of UAVs deployed.

This sudden progress of UAVs and, more importantly, their commercial application in an ever-larger spectrum of scenarios increases the need for safer and more reliable algorithms [13–16]. The UAVs are a forefront of upheaval developments of sensing technologies (e.g., thermal, multispectral, and hyperspectral) in multiple areas that may change parts of society by creating new solutions and applications [17–22].

The increased safety in the UAV operation offered by the proposed solution allows new and interesting usage scenarios, such as urban event filming. A novel Collision Avoidance algorithm is proposed, which utilizes a Neural Network Pipeline (NNP) that consists of a Convolutional Neural Network (CNN) to extract features from video frames, a Recursive

Neural Network (RNN) that takes advantage of the video temporal characteristics, and feeds a Feed-forward Neural Network (FNN) that is capable of estimating if there is an incoming collision. In parallel, an algorithm based on optical flow and flow agglomeration uses the latest two frames to calculate the closer object trajectory. The output of this algorithm is only taken into consideration whenever the NNP detects an incoming collision. Having the closest object trajectory, it is trivial to calculate a reactive escape trajectory.

To prevent a collision with a dynamic obstacle (such as an animal) or an incoming object (such as a thrown ball), a UAV needs to detect them as fast as possible and execute a safe maneuver to avoid them. The higher the relative speed between the UAV and the object, the more critical the role of perception latency becomes [23–25].

Researchers elaborated Collision-Avoidance Vector Field (CAVF) in the presence of static and moving obstacles [26] but did not tackle the problems of estimating the objects speed in real scenarios, where the sensors generate not so accurate data, and the objects are not well defined. Some solutions using deep reinforced learning are starting to appear in literature [27–29], which explore Neural Networks for autonomous navigation, but still only show a low Technology Readiness Level (TRL) and are not yet ready for industrial applications.

Compared to collision avoidance algorithms for static objects, the avoidance of dynamic objects have not yet been that much explored, since the task is much harder [5,30]. There are some works, such as the one from Poiesi and Cavallaro where multiple image processing algorithms that estimate the time of collision of incoming objects [31] are explored. The detection is accurate, but the algorithm takes more than 10 s to process each frame, making the solution not applicable in real time scenarios with state-of-the-art hardware. In addition, Faland et al. [32] delved into the event cameras to generate a computing efficient sensing pipeline, that was capable of avoiding a ball thrown towards a quad-copter at speeds up to 9 m/s similar to the work done in Reference [24]. To have a comparable test-bed, this use-case was considered for testing our algorithm, requiring the creation of a novel dataset with over 600 videos of different individuals throwing balls at a UAV.

The contributions of this article are then threefold:

1. the development of an efficient and simple but robust software architecture for reacting and avoiding collisions with static or dynamic obstacles that are not known beforehand;
2. proposal of a dataset of different individuals throwing balls at a UAV;
3. and collision Avoidance Algorithm that uses an NNP for predicting collisions, and an Object Trajectory Estimation (OTE) algorithm using Optical Flow.

The rest of this paper is structured in the following sections: Section 2 presents the proposed solutions, being fully explained to facilitate replicability. Initially, a high level overview of the Framework for Collision avoidance for autonomous UAVs is provided in Section 2.1, and the main novelties of this architecture is the collision module, which is detailed, respectively, in Section 2.2, divided in the NNP and the Flows processing. Section 3 describes the experimental testing of the solution, followed by Section 4 that analyzes the data collected, which validates our approach and implementation. The paper presents its main conclusions and possible future research work in Section 5.

## 2. Materials and Methods

To tackle the collision avoidance task in UAVs, the architecture in Figure 1 is proposed. Initially, a generic view of all the modules and technologies is provided. Afterward, a detailed explanation is provided for the avoidance of incoming objects.

### *2.1. Collision Avoidance Framework for Autonomous Uavs*

The communication between blocks is executed throughout ROS topics and services using the publisher/subscriber paradigm [33,34]. The cloud between the user and the UAV can be performed through the Wi-Fi, Zigbee and/or 4G protocols. In this way, the user is able to remotely communicate with/control the UAV, regardless of the distance. An architecture of the main modules is depicted in Figure 1.

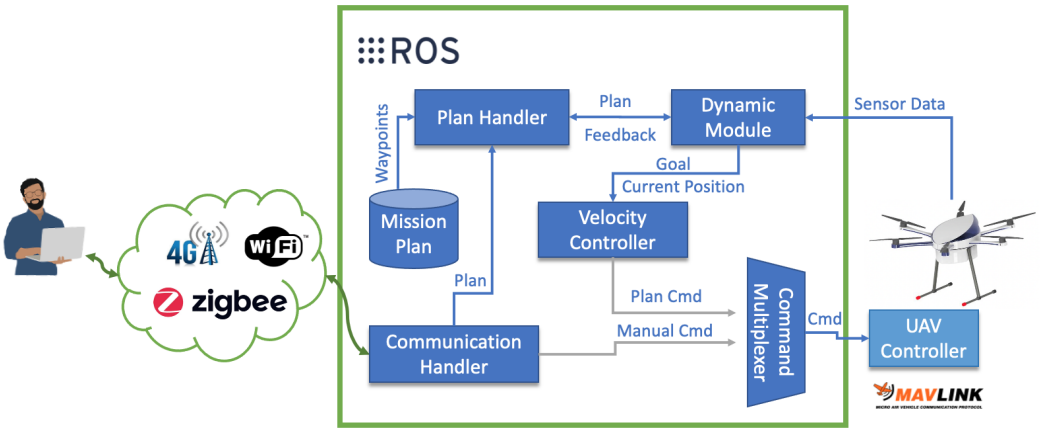



**Figure 1.** Architecture of the proposed framework for safer UAVs.

The proposed architecture can be divided into five main blocks:

1. Communication Handler: The Communication Handler block is responsible for maintaining interoperability between the user and the UAV. It is also responsible for triggering the pre-saved UAV mission through an activation topic.
2. Plan Handler: This block is responsible for sending each waypoint of the complete mission to the Positioning Module block through a custom service in order to increase the security of communication and the entire system pipeline [35]. In this custom service, the Positioning block asks the Plan Handler block for the next point of the mission to be reached. In turn, the Plan Handler block returns the next point, where they contain the local coordinates of the intended destination.
3. Dynamic Module: This module computes possible dynamic object collisions. Through the camera, the inertial sensors, and the algorithm proposed in Section 2.2, it is possible to detect and avoid dynamic objects. Figure 2 presents the connections required for this module works.
   It is possible to observe in Figure 2 that there are 2 distinct processes: NNP for the static and dynamic obstacles prediction and detection; Clustering technique for grouping different objects in the same image and its, respectively, 2D movement direction to know the UAV escape trajectory.
   Through the inertial sensor, it is possible to know the UAV position and how to avoid when the obstacle avoidance algorithm is activated. In case the algorithm is not activated, the pre-defined mission by the user is carried out, through the Plan Handler module. The UAV position and the desired destination are then sent to the Velocity Controller block that navigates the UAV to the desired destination.
4. Velocity Controller: The Velocity Controller block calculates the velocity required to reach the desired destination (with the mavros package [36]) using the inputs from the Positioning Module block and the Dynamic Module. This controller extends a proportional–integral–derivative (PID) controllers, where the variables change depending on the type of UAV, and the UAV's velocity calculation on the three axes were based on Reference [37], where:

$$eP^{(t)} = gP^{(t)} - cP^{(t)}, \quad (1)$$

and

$$eD^{(t)} = ||(eP^{(t)})||, \tag{2}$$

where $eP^{(t)}$ represents the error position, $gP^{(t)}$ the goal position, $cP^{(t)}$ the current position at time instant $t$, and $eD^{(t)}$ is the distance error position $eP^{(t)}$.
With Equations (1) and (2), it is possible to normalize the error, as shown in Equation (3).

$$eN^{(t)} = \frac{eP^{(t)}}{eD^{(t)}}, \tag{3}$$

where $eN^{(t)}$ is the error normalized.
If the distance is lower than a certain threshold, $\tau$ (in this work, the threshold value value is set to $\tau = 4$ m), Equation (4) is activated.

$$vP^{(t)} = eP^{(t)} \cdot \left(\frac{eD^{(t)}}{\tau}\right)^{SF}, \tag{4}$$

where $vP^{(t)}$ is the velocity vector, and $SF$ is the Smooth Factor (the $SF$ was set to 2 [37]).
If the distance is higher than 4 m (threshold), Equation (5) is then used.

$$vP^{(t)} = eN^{(t)} \cdot PMV. \tag{5}$$

In Equation (5), $PMV$ is the Param Max Velocity and is equal to 2.
In this way, it is allowed to dynamically vary the UAV speed depending on the UAV distance in relation to the desired destination without any sudden changes regarding the UAV's acceleration;

5. Command Multiplexer: The Command Multiplexer (CM) block subscribes to a list of topics, which are publishing commands and multiplexes them according to a priority criteria. The input with the highest priority controls the UAV by mavros package [36] with the mavlink protocol [38], becoming the active controller.

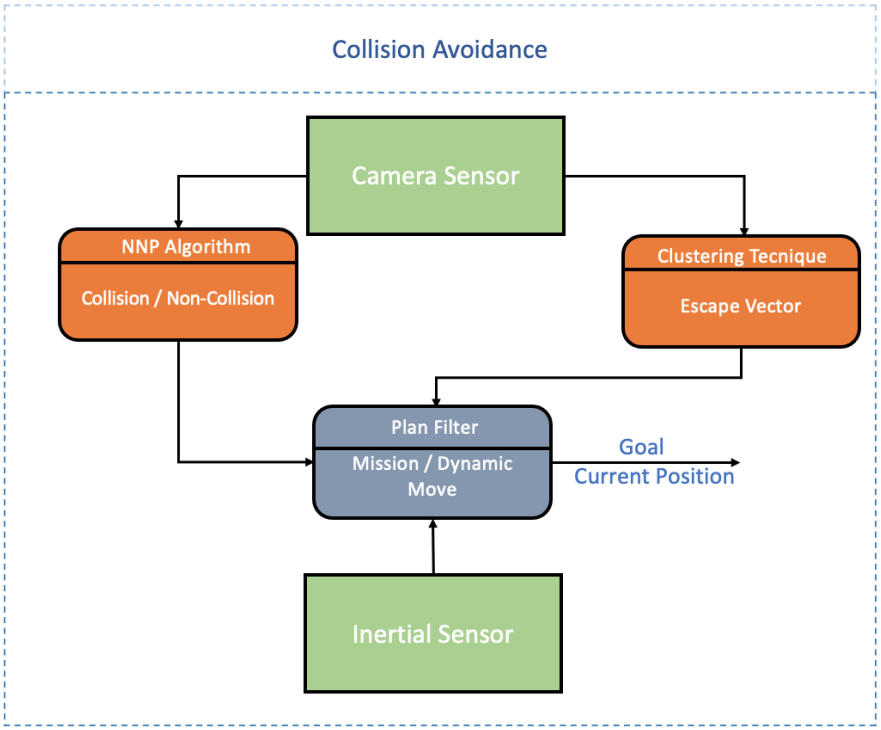



**Figure 2.** The proposed collision avoidance approach for autonomous missions.

### *2.2. The Proposed Collision Avoidance Algorithm*

To detect and avoid a possible collision with moving objects, a combination of two novel algorithms is proposed, that should be executed in parallel threads to boost performance. The first uses a NNP that predicts if there is an incoming collision. The second analyzes the pixel flows and applies clustering techniques estimate the image objects motion, allowing the calculation of the dynamic object trajectory. Only when NNP detects a collision are the results of the objects flow considered. A pseudo-code example is shown in Algorithm 1.

Whenever a possible collision is detected, the algorithm can use the latest available escape trajectory to dodge the incoming object. If the processing time of the NNP is greater than 1/fps, an additional thread must be added, which keeps reading the frames in parallel and updating the frame buffer that is used by the NNP. This thread makes sure that both the NNP and the OTE are using the latest frames, with no lost frames, or variable sequencing.

**Algorithm 1:** Proposed Algorithm for collision avoidance with moving objects.

```
# message publisher
reactiveCmdPub = ros.Publisher()

# last known escape vector
escapeVector = {x: 0.0, y: 0.0, z: 0.0}

# OAF algorithm will constantly update the escapeVector var
opticalFlowThread = threading.Thread(target=OAF)

# Callback for Hybrid Collision Avoidance function
# it should be called whenever a new frame is obtained
def hca(videoFrame):

# Use the DCA algorithm to detect collisions
if(dcaProcessFrame(videoFrame)):
reactiveCmdPub.pub(escapeVector)
```

Section 2.2.1 will describe the NNP model that was developed in this article, and Section 2.2.2 will address the algorithm developed in order to agglomerate different incoming objects detected in a given image.

#### 2.2.1. Neural Network Pipeline

A UAV must detect and perform a safe maneuver to avoid a collision with a incoming obstacle (such as an animal) or an incoming target (such as a tossed ball). Researchers [23–25] have defined perception latency as the time used to perceive the environment and process the captured data to generate control commands. This is a key metric when designing collision avoidance algorithms. The higher the relative speed between the UAV and object, the more critical the role of perception latency becomes.

This article proposes an innovative approach that makes use of a NNP. The task was divided into three sections to make it easier to solve. The first block is called Feature Extraction (FE), and it uses a CNN to generate feature vectors for video frames. The second block uses RNNs and the input of several $SEQ$ feature vectors to manage the video temporal information (stream). Finally, the third block receives the outcome of the previous RNN and employs a FNN to generate a decision. Figure 3 depicts the proposed architecture. These blocks will be discussed in more detail in the following subsections. Implementations, visualization functions, and additional details are available at https://github.com/dario-pedro/uav-collision-avoidance/tree/master/train-models (accessed on 20 May 2021).

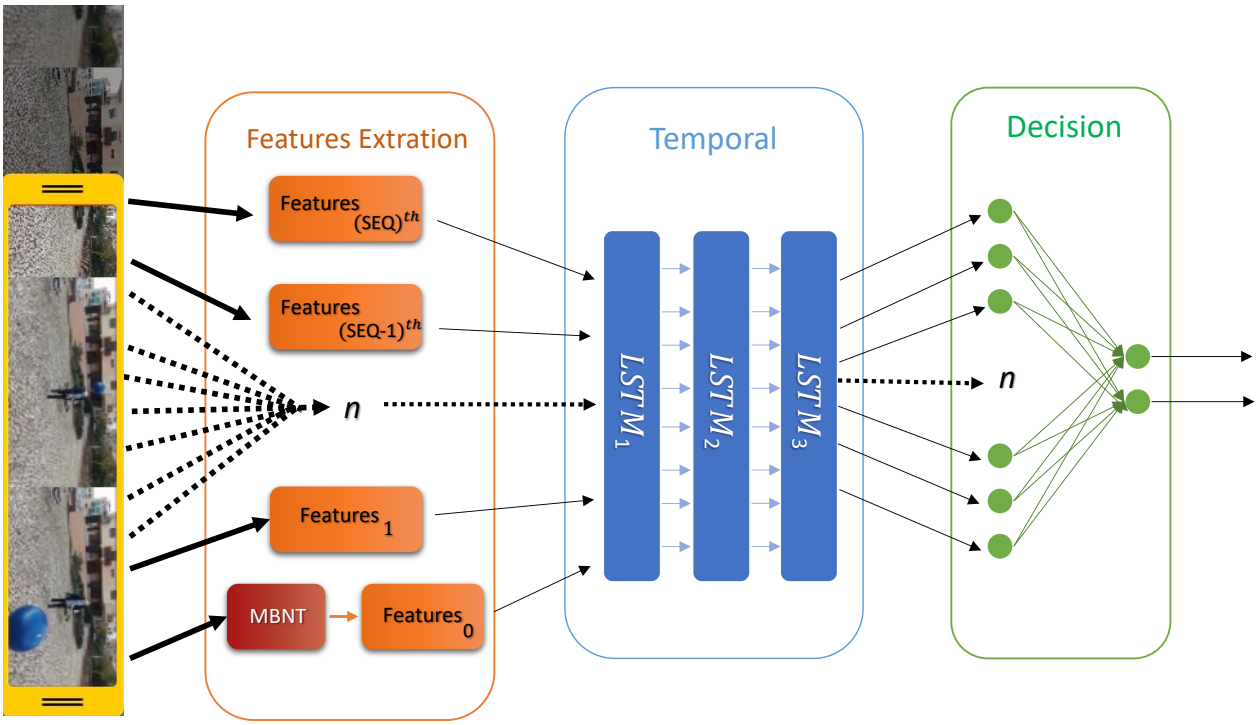



**Figure 3.** Proposed neural network pipeline for the detection of possible collisions.

Feature Extraction

The process of FE can be summarized in processing each frame with a CNN, which produces a vector, that can be interpreted as the frame key features that will ultimately be used by the NNP to detect a collision. Due to performance reasons, a MobileNetV2 (MNV2) [39] was selected. This CNN's model is built on an inverted residual configuration, under which the standard residual block's input and output are thin bottleneck layers, in contrast to residual models that use extended representations in the input and lightweight depth-wise convolutions to process features in the middle expansion layer [39]. This allows it to achieve the best trade-off between accuracy and computation for a low-power processor as the one present in UAVs [39–41]. For the input, it requires a $224 \times 224 \times 3$ matrix, and, after processing, it generates a $7 \times 7 \times 1280$ output matrix, which is converted into a 1280 vector by applying a 2D Global Average Pooling [42].

Temporal Correlation and Decision

A RNN is used to achieve the temporal association of the function data derived from each frame. A 3-depth blocks Long Short Term Memory (LSTM) [43] architecture is proposed in this article, which receives a series $\varphi$ of 25 input vectors, representing approximately 1 s of video at a frame rate of 25 (the average video framerate of the selected dataset, the ColANet). The first layer contains eight LSTMs, while the second and third layers each contain two LSTMs. Dropout and batch normalization are applied after the first three layers. Furthermore, the final RNN layer is linked to a Feed-forward Neural Network with four neurons, which is then linked to two output neurons.

In a real-world use-case, the architecture is implemented using a sliding window technique, with the function queue still including the last 25 feature vectors and being fed into the RNN. When a new video frame becomes available, the FE processes it and adds the new feature vector to the features queue by moving the previous values. Furthermore, the resulting decision of the last frame is the consequence of the RNN and FNN for a group of 25 function vector arrays. Algorithm 2 describes the steps required to process a new video frame, assuming that all neural networks have already been loaded.

**Algorithm 2:** Neural Network Pipeline—processing the latest video frame.

```
SEQ_LEN = 25
features_queue = deque(maxlen=SEQ_LEN) # Double-ended queue

def dcaProcessFrame(videoFrame):
# Resize image to cnn input size
img = video_frame.resize(224,224,3)

# ML libs predict functions outputs arrays
cnn_pred = cnn_model.predict(img)[0]

# Shift add the image features to the features queue
features_queue.append(cnn_pred)

# Check if enough images have been seen
if(len(features_queue) >= SEQ_LEN ):
rnn_pred = rnn_model.predict(features_queue)[0]
return decision_model.predict(rnn_pred)[0] # return result
else:
return 0 # return no collision
```

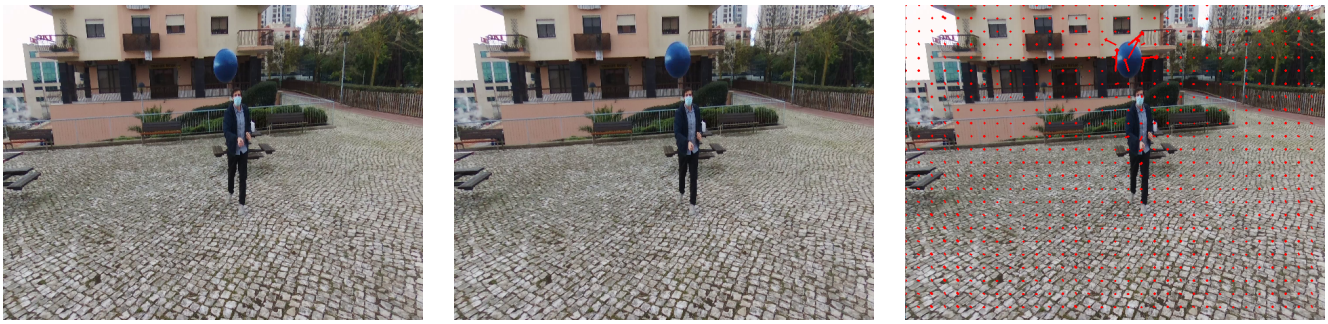
Comparing solutions, such as conv3d [44], which apply convolutions to a 3D space, the processing is streamlined and, therefore, highly optimized. Just the last frame has to be interpreted by the CNN and the output features inserted into a queue that is forwarded to the RNN. Following that, the RNN and FNN are executed, which will provide a prediction.

#### 2.2.2. Object Trajectory Estimation

The NNP presented is capable of detecting collisions or estimating escape trajectories. But, for practical scenarios, some federal organizations fear deploying algorithms that are only based on Neural Network (NN) architectures because they lack a results explanation [45]. Furthermore, if the task of the AI algorithm is simplified to the collision prediction, it increases its performance. For this reason, an Object Motion Estimator (OME) that utilizes Optical Flow (OF), and that can run on a parallel thread, was developed.

The OF is defined as the change of light in the image, e.g., the retina or the camera sensor, associated with the motion of the scene relative to the eyeball or the camera. In a bio-inspired sense, shifts in the light captured by the retina result in a movement perception of the objects projected onto the retina. In the technical context of computer vision, a set of video frames contain the movement of the observer and the environment combined.

NVIDIA Turing GPUs include dedicated hardware for OF computing. This dedicated hardware uses sophisticated algorithms to generate highly accurate flow vectors that are robust for frame-to-frame variations in intensity and track true object motion. Computation is significantly faster than other methods with comparable accuracy. The NVIDIA library for the pyramidal version of Lucas-Kanade method, which computes the optical flow vectors for a sparse feature set, was used to estimate the objects movement. The result of this algorithm on two frames at $t-1$ and $t$ is illustrated at Figure 4. In Figure 4c, it is possible to observe the magnitude and direction of the flows matrix, each represented by a red arrow.



(**a**) (**b**) (**c**)

**Figure 4.** Optical Flow result from frames (**a**) First Image at $t-1$ time; (**b**) Second Image at $t$ time; (**c**) Optical Flow result between images 4a and 4b.

Calculating the OF of an image generates a matrix of flows that can be used to estimate the objects flow. Nevertheless, some flows are outliers and others are tracks of the object and parts of the background that were covered and became unveiled. In literature, there are multiple algorithms capable of clustering data. Nevertheless, none of them are tailored for the concrete case of low-processing, highly variable, objects flows. For this reason, a novel algorithm that filters and agglomerates flows in groups, outputting an aggregated flow result, with the goal of obtaining the closest object true flow is proposed. This algorithm is entitled Optical Flow Clustering. The most known clustering techniques were also implemented in order to benchmark the proposed algorithm.

To facilitate the comparison between metrics and results, the algorithms were divided by: the feature vectors representation and normalization of the flow data; appropriate distance measures and data reduction; and data clustering techniques.

Flow Vectors

To obtain the feature space $\chi$, a four-dimensional vector space with $N$ feature vectors $f = \begin{pmatrix} x & y & u & v \end{pmatrix}^T$ is considered, where $p = \begin{pmatrix} x & y \end{pmatrix}^T$ are image pixel location coordinates, and $\psi = \begin{pmatrix} u & v \end{pmatrix}^T$ are velocity vectors.

The $\Re(v)$ is the magnitude of a vector, and $\Theta(v) = \angle(v_i, v_j)$ is the angle between any two vectors $v_i$ and $v_j$, with $1 \leq i, j < N$. The $N$ function vectors are drawn at random from dense optical flow fields obtained for the measurement duration using a real-time variational method recently published [46]. Flow vectors with the smallest magnitude are discarded (by default 10%). Random sampling is used for statistical purposes, so clustering can be processed in miliseconds. We omit time details from the function vectors since the examined video sequences are comparatively small. By subtracting the average and dividing by the standard deviation, we normalize the image position coordinates $x$ and $y$, as well as the velocity components $u$ and $v$.

Flow Distances and Dimension Reduction

Three distance measures are defined: $D(i,j) := D(f_i, f_j)$ between any two feature vectors, with $1 \leq i, j < N$ :

- Euclidian: $D_E(i,j) = \sqrt{\sum_K^k (k_i - k_j)^2}$;
- Manhattan: $D_{Mt}(i,j) = \sum_K^k |k_i - k_j|$; and
- Mahanalobis: $D_{Mh}(i,j) = \sum_K^k (k_i - k_j)^{\Sigma^{-1}} (k_i - k_j)^T$, where $\Sigma$ is the covariance matrix between the components of the feature vectors.

Dimension reduction assists in data compression and reduces computation time. It also aids in the removal of some unnecessary functions. Furthermore, it reduces the time needed for clustering computation. For this reason, some dimension reduction techniques were also integrated:

- Isomap [47] is a low-dimensional embedding approach that is commonly used to compute a quasi-isometric, low-dimensional embedding of a series of high-dimensional data points. Centered on a rough approximation of each data point's neighbors on the manifold, the algorithm provides a straightforward procedure for estimating the intrinsic geometry of a data manifold. Isomap is highly efficient and can be applied to a wide variety of data sources and dimensionalities.
- Multidimensional Scaling (MDS) [48–50] is a technique for displaying the degree of resemblance between particular cases in a dataset. MDS is a method for converting the information about the pairwise 'distances' among a collection of vectors into a structure of points mapped into an abstract Cartesian space.
- T-distributed Stochastic Neighbor Embedding (t-SNE) [51,52] is a mathematical method for visualizing high-dimensional data by assigning a position to each datapoint on a two or three-dimensional map. Its foundation is Stochastic Neighbor Embedding. It is a nonlinear dimensionality reduction technique that is well-suited for embedding high-dimensional data for visualization in a two- or three-dimensional low-dimensional space. It models each high-dimensional object by a two- or three-dimensional point in such a way that identical objects are modeled by neighboring points and dissimilar objects are modeled by distant points with a high probability.

Flow Clustering

In order to generate the region of interested of the incoming object, the following clustering methods have been implemented:

- Kmeans [53] is a vector quantization clustering technique that attempts to divide $n$ observations into $c$ clusters, with each observation belonging to the cluster with the closest mean (cluster centers or cluster centroid), which serves as the cluster's prototype. As a consequence, the data space is partitioned into Voronoi cells [54].
- Agglomerative Ward (AW) [55] is a Agglomerative Clustering technique that recursively merges the pair of clusters that minimally increase the wards distance criterion. Ward suggested a general agglomerative hierarchical clustering procedure in which the optimal value of an objective function is used to pick the pair of clusters to merge at each node.
- Agglomerative Average (AA) [56] is a clustering technique that recursively merges pairs of clusters, ordered by by the minimum average distance criterion, which is the average of the distances between each observation.

In addition to the state-of-the-art clustering techniques, a novel algorithm, entitled Optical Flow Clustering was developed, which is finely tailored for the collision detection. To process the OFC, initially it is necessary to calculate the image normalization factor (Equation (6)), $\varrho$:

$$\varrho = W^2 + H^2, \tag{6}$$

where $W$ is the image width, and $H$ the image height. Then, the OF matrix is obtained, by computing the flow between $frame_{t-1}$ and $frame_t$. The resulting flows need to be filtered to reduce noise and ensure that only meaningful flows are considered for agglomeration. The value should be normalized by $\varrho$ to compare to the flow threshold, $\phi_T$. A standard value for $\phi_T$ is 1%, varying mostly with the camera stabilization (which induces noise). This filtering can be obtain by Equation (7).

$$\sqrt{\frac{x_w^2 + y_h^2}{\varrho}} \geq \phi_T. \tag{7}$$

The next step is an iterative procedure. It starts by considering two flows $f_0$ and $f_1$, from which the current position $P_{r0} = (x_0, y_0)$ and $P_{r1} = (x_1, y_1)$ is obtained, along with the flow ending position $(x_{0d}, y_{0d})$ and $(x_{1d}, y_{1d})$, which is $(x_n + f_{nx}, y_n + f_{ny})$. An example of these flows and positions is illustrated in Figure 5.

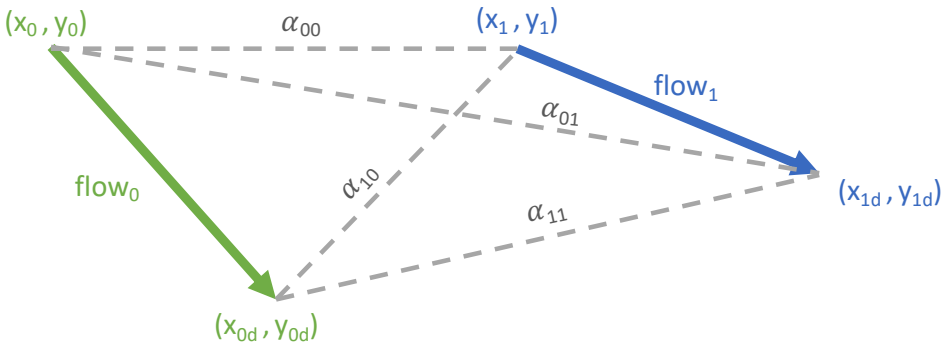



**Figure 5.** The $\alpha$ distances obtained from flows magnitude and directions vectors.

Using these positions, it is possible to calculate the $\alpha$ distances:

$$\begin{bmatrix} \alpha_{00} & \alpha_{01} \\ \alpha_{10} & \alpha_{11} \end{bmatrix} = \begin{bmatrix} \sqrt{\frac{(y_0-y_1)^2+(x_0-x_1)^2}{\varrho}} & \sqrt{\frac{(y_0-y_{1d})^2+(x_0-x_{1d})^2}{\varrho}} \\ \sqrt{\frac{(y_{0d}-y_1)^2+(x_{0d}-x_1)^2}{\varrho}} & \sqrt{\frac{(y_{0d}-y_{1d})^2+(x_{0d}-x_{1d})^2}{\varrho}} \end{bmatrix}. \tag{8}$$

The $\alpha$ distances presented in Equation (8) are used to verify if two flows can be merged, by comparing their values with $\alpha_{threshold}$. If the calculated value is below the $\alpha_{threshold}$, it is a valid flow to be merged. Whenever the values of the flows increase greatly and share the same direction, all the $\alpha$ distances might be larger then the $\alpha_{threshold}$, but still represent a flow from the same object. For this reason, it is important to calculate the distance of the centers of both flows $D_c$ and the radius of the enclosing circumferences $R_{fn}$ (Equations (9) and (10)):

$$D_c = \sqrt{\frac{(|y_0 - y_{0d}| - |y_1 - y_{1d}|)^2 + (|x_0 - x_{0d}| - |x_1 - x_{1d}|)^2}{2}}, \tag{9}$$

and

$$\begin{cases} R_{f0} = \sqrt{\frac{(y_0-y_{0d})^2+(x_0-x_{0d})^2}{2}} \\ R_{f1} = \sqrt{\frac{(y_1-y_{1d})^2+(x_1-x_{1d})^2}{2}} \end{cases}. \tag{10}$$

Figure 6 represents the intersection of the enclosing circumferences, which can be verified by the condition $R_{f0} > D_c + R_{f1}$ *or* $R_{f1} > D_c + R_{f0}$.

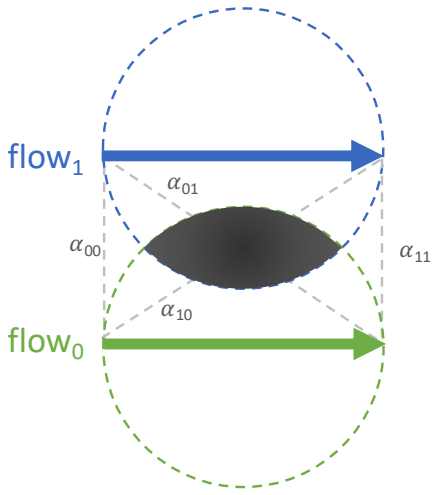



**Figure 6.** Intersection of the enclosing circumferences generated by the obtained flows.

Whenever the $\alpha$ distance or the intersection of the enclosing circumferences is verified, calculate $\{y_{min}, x_{min}; y_{max}, x_{max}\}$ of the considered positions and merge Flows $f_0$ and $f_1$. The merge can be obtained by Equations (11) and (12):

$$P_{r0}(x,y) = \frac{P_{r0} \cdot \nu_{f0} + P_{r1} \cdot \nu_{f1}}{\nu_{f0} + \nu_{f1}}, \tag{11}$$

$$F_{r0}(x,y) = \P_{r1} - \P_{r0}. \tag{12}$$

Then, the $f_1$ is removed from the flow list. For a given region group, this processed is iterated considering the next $P_{r1}$ the left value of the list. When no flows can be agglomerated, that flow is stored and the next two flows are considered. This process is executed through the entire list of flows, and only stops whenever no flows are merged in a full search. The final result are regions containing the group of flows, and the cumulative flow values at the center of the regions. Figure 7 represents the result of the OFC on the flows processed and previously represented in Figure 4.

**Figure 7.** Optical Agglomerated Flow enclosing circumferences.

The output of the OFC are regions, that can be considered as moving objects. The incoming colliding object is considered to be region with the bigger area (supposedly closer to the camera). For example, on Figure 8, the hand of the person throwing the ball towards the UAV has produced a region with flows, which is smaller than the flow produced by the ball, and that needs to be discarded. By the incoming colliding object flow is possible to calculate an escape trajectory $v$, which is the perpendicular vector $v^{\perp}$, giving preference to rising solutions. Note that a perpendicular 2D vector $v^{\perp}$ always has two solutions, (90° and −90°). For a UAV, it is usually safer to go up; therefore, dodging objects by rising the UAV is considered safer). The OFC algorithm is depicted in pseudo-code in Algorithm 3.

**Figure 8.** Optical Agglomerated Flow with two regions.

**Algorithm 3:** Optical Flow Clustering algorithm.

```
# threshold to filter the flows
flowThreshold = 1
# N value for normalize the flows with image width and image height
N = math.sqrt(pow(w, 2) + pow(h, 2))
# threshold to filter the alpha distances
distanceThreshold = 15
# Callback for Hybrid Collision Avoidance function
# it should be called whenever a new frame is obtained
def OAF(frame1, frame2):
# obtain the optical flow from the 2 frames
flows = cv2.cuda_OpticalFlow.calc(frame1, frame2)
# filter meaningful flows
flows = filterFlows(flows, flowThreshold)
aggregating = True # control variable
regions = [] # object regions
while aggregating: # stop when there is no flows to merge
aggregating = False
for i in len(flows)-1:
for j in len(flows)-1:
if(i != j): # do not compare with self
# calculate the flows
alphas = calculateAlphas(flows[i], flows[j])
# radius and centers
rac = calculateRaC(flows[i], flows[j])
if(validAggregate(alphas, rac, distanceThreshold)):
# force a full scan
aggregating = True
# merge flows into flow[j]
mergeFlows(flows[j], flows[i)
if flows[j] not in regions:
# new region found - append it
regions.append(flows[j])

del flows[i] # remove the merged flow
break # go back to the first for cycle
```

## 3. Results

This section validates the performance of the proposed algorithm through numerical results and is organized in three parts. The first one presents the training results of the NNP, the second showcases the real environment results to assess the performance of the proposed algorithm from both accuracy and operational time point of views. The third part will compare the obtained results with the SoA methods.

### *3.1. NNP Training and Results*

To train the proposed NNP, the creation of a new dataset focused on the selected testing scenario, the avoidance of a thrown ball, was required. The techniques proposed in ColANet [57] were used as guidelines, and the new dataset was made available at https://ballnet.qa.pdmfc.com/ (accessed on 20 May 2021). This dataset has 600 videos, which represent a total of 20,000 images. Frameworks for machine learning Tensorflow and Keras were used to aid in the creation and training of the models of the NNP [58].

At first, the MNV2 model was created in Tensorflow using the transfer learning method [59]. The weights were pre-trained with the ImageNet dataset, which contains 1.4 million images and 1000 categories of web images [60]. ImageNet is a fairly diverse dataset, containing categories, such as plane and eagle; however, this information facilitates the FE processes and transfers the general world perception.

First, the MNV2 layer used for FE is determined. The final classification layer (on top, like most models, which go from bottom to top) is useless. Typically, it is a standard

practice to use the final layer before the last flattening layer. This layer is referred to as the 'bottleneck layer'. As compared to the final/top layer, bottleneck features maintain a lot of generality. This can be accomplished by loading a network without the classification layers at the end, which is suitable for FE.

Secondly, all layers are frozen before compiling the model, avoiding weights from changing during training. Then, a classification block is applied, which consists of a global average pooling 2D layer to convert the features to a single 1280-element vector per image and a dense layer to convert these features to a single prediction per image. Since this prediction would be viewed as a raw prediction value, no activation mechanism is needed. Positive numbers indicate a collision, while negative numbers indicate no collision.

Using a binary cross-entropy loss and an Adam optimizer [61] with a $1 \times 10^{-4}$ learning rate and a $1 \times 10^{-6}$ decay rate, the final classification layer was trained to have some knowledge of the objective target. The logistic regression loss can be represented as Equation (13) for the classification problem under consideration.

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)], \tag{13}$$

where $w$ denotes the model parameters (weights), $N$ denotes the number of images, $y_i$ denotes the target label, and $\hat{y}_i$ denotes the projected label. Equation (14) gives the accuracy metric:

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N} 1(\hat{y}_i = y_i). \tag{14}$$

The training results of the added classifier are seen in Figure 9 for the first 20 epochs (before fine-tuning). Using only single frame knowledge, the model predicted collisions with 54.4% accuracy at the end of the 20 epochs (validation accuracy). Following that, a refined version of the MNV2 base model was trained. To do this, all layers were unfrozen. It is vital to emphasize that the first step is needed, since, if a randomly initialized classifier is applied on top of a pre-trained model, and all layers are jointly trained, the magnitude of the gradient updates will be too high (due to the classifier's random weights) and the pre-trained model will forget what it has learned (the transferred knowledge). The fine-tuned FE's training results are the last 20 epochs of Figure 9 and obtained a final validation precision of 66.8%. The disparity between training and validation began to increase on the final epochs, indicating the beginning of over-fitting, and no further epochs were trained as a result.

The MNV2 network has been fine-tuned to produce a model that is highly oriented to the collision classification challenge. On the one hand, this is positive because it helps the CNN to emphasize certain features, but, on the other hand, it is negative because it does not generalize well (potentially contributing to overfitting data) and gives more weight to features present in the dataset. As a result, the RNN and FNN blocks' training will be achieved in just two iterations. The first iteration employs the MNV2 with ImageNet pre-trained weights. The second iteration employs the MNV2 with ImageNet pre-trained weights that have been fine-tuned in the novel dataset.

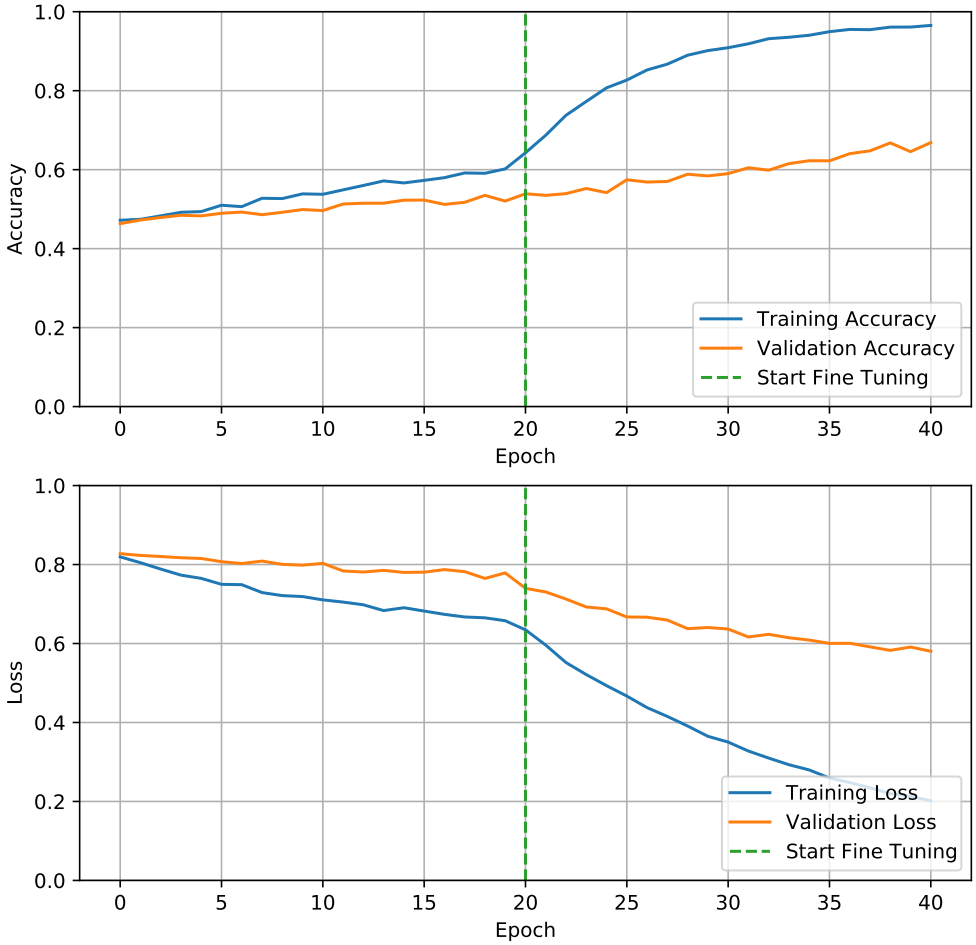



**Figure 9.** Training Feature Extraction based on MobileNetV2 model. On the first 20 epochs, only output neurons are trained. Afterwards, the entire model is fine tuned.

Initially, the RNN+FNN blocks are trained using CNN resulting data that has not been fine-tuned with dataset data. Some restrictions must be applied to the data in order to ready it for the second block:

1. The input data must be a SEQ-length sequence. In this article, a value of 25 was used, but any value between 20 and 50 produced comparable results.
2. The sequences produced must only contain frames from a single video. Working with video data on GPUs is not an easy process, and creating video sequences adds another layer of complexity. The model perceives the dataset as a continuous stream of data, and this constraint must be applied to prevent the model from learning jumps between videos (false knowledge).
3. The goal for the whole series is the last frame target label.

The Adam optimizer [61] was used, with a learning rate of $1 \times 10^{-4}$ and a decay rate of $1 \times 10^{-6}$. Figure 10 shows the training results of the RNN with the FNN classification layer with moved FE weights and fine-tuned FE.

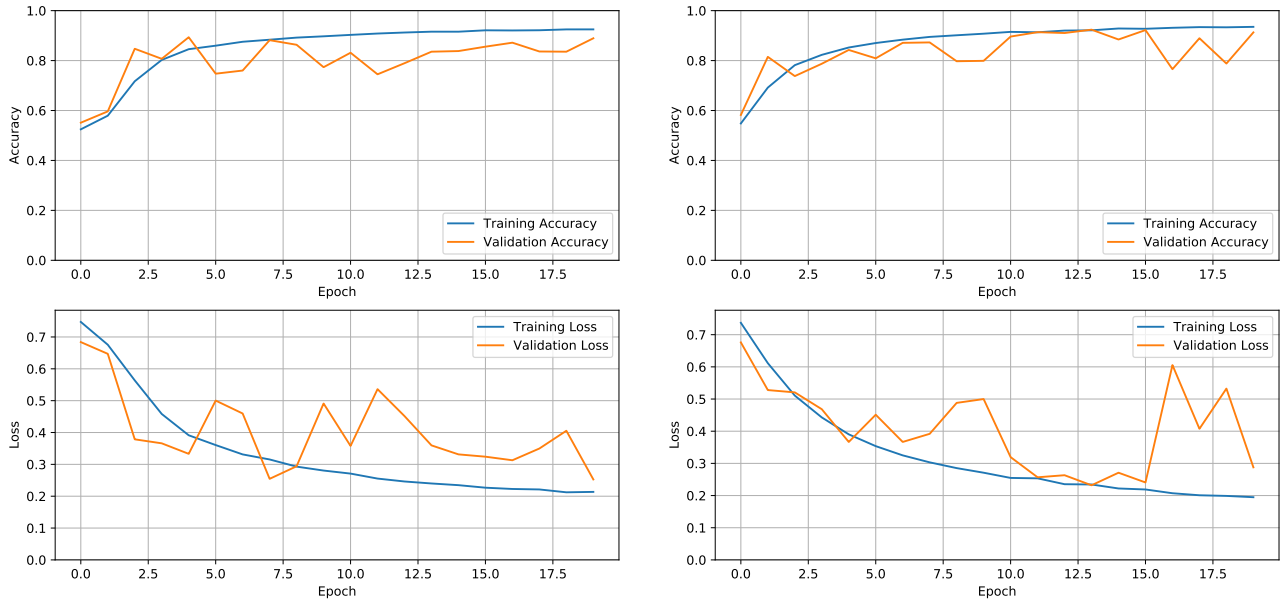



**Figure 10.** Training evolution graph of the NNP models: (**a**) First iteration—Using FE with the default MNV2 weights (ImageNet weights); (**b**) Second iteration—Using FE fine-tuned weights.

### *3.2. Real Environment*

The avoidance of a thrown ball was tested to validate the algorithm in a similar use-case to previous research works [32]. A Parrot Bebop 2 was connected to a Legion with a 2060 GPU, running the proposed framework.

At first, the trained NNP weights were used, but the results were worse then expected. After some troubleshooting, the authors realized that by using the recorded videos, a constant framerate of approximately 29 fps was obtained. However, when working with the livestreaming video from the UAV, a framerate high variance was experienced, oscillating between 5 to 30 fps. On top of that, the compression algorithm used by the Parrot Bebop in livestream mode also reduces the video quality, creation yet another major difference against the trained NNP. The transmission delay is also an issue, and, for simplification, it will be left out of this paper.

To solve this issue, a set of image augmentation techniques were applied to the dataset. It consisted in randomly exposing the model to videos at variable framerate (by dropping frames), compressed frames (constant within the video, variable per epoch), and applied the most traditional augmentations, such as rotation, translation, and zoom (yet again, constant per video). After training and deploying this new model, a normal behavior was obtained. The NNP result is far from perfect, but this is due to the fact that the score is measured at the frame level. On the testing results, the NNP often detects a collision a few frames before or after the ideal moment annotated on the dataset, lowering the score. This is not critical, as long as the detection is obtained at a moment previously enough for the dodge routine.

Some of the latest UAVs in market (e.g., Skydio 2, HEIFU [62,63]) already have Nvidia Single Board Computer (SBC)s, being capable of running such algorithms directly on the aircraft, therefore being capable of running the proposed architecture directly on the UAV. The NNP and OME were integrated in a Jetson Nano (SBC) used by HEIFU), which managed to run the entire algorithm pipeline in an average of 0.18 s, demonstrating that it is a viable option for SoA UAVs.

In order to study the proposed OME algorithm, 8 frames from 8 different videos (total of 64 frames) were selected. Figure 11 illustrates these frames, which were manually annotated with a red mask on the ball, allowing the creation of ground truth mask by the filtering by color. The NNP were correctly outputing collision for all the selected frames;

therefore, it is possible to evaluate the performance of the OME algorithm (the most critical part of the dodging trajectory estimation). The results of the overall approach are further analyzed on the next subsection.

**Figure 11.** Set of 64 ground truth frames for the OBE results discussion.

## 4. Discussion

The NNP appers to be a vilable solution for the detection of incoming objects. Table 1 summarizes the outcomes of the proposed models pipelines. It is possible to infer that it is a viable solution for the detection of incoming collisions, but more research, datasets, and testing are needed. On unseen data, the trained MNV2 achieved an accuracy of 66.8%, while the complete NNP using temporal features from an untrained MNV2 achieved an accuracy of over 89%. This confirms the theory that temporal information is important in the collision detection problem (or possibly in any video-related classification problem).

**Table 1.** Collision avoidance trained models' results comparison.

| Metrics | $FE_1$ MNV2 | Fine-Tuned MNV2 | NNP w/ MNV2 | NNP w/ Fine-Tuned MNV2 |
|---|---|---|---|---|
| **Training Accuracy** | 64.6% | 97.4% | 92.6% | 93.4% |
| **Validation Accuracy** | 54.4% | 66.8% | 89.4% | 91.4% |

Fine-tuning the MNV2 improved the results of the NNP, but it is a slight trade-off between generalization and dataset performance. The presented dataset is relatively small and narrow, with a limited range of environments and variability. Because of the model's

proclivity to overfit, this makes preparation more difficult. The models can be further generalize and show better results as the amount of available UAVs datasets grows.

Futhermore, the OME is capable of calculating escape trajectories for the closest detected object. For each frame illustrated in Figure 11, the previous frame and the current frame were fed into the OME algorithm, that outputted a region for the incoming object. Using this output and the ground truth mask $GT$, it is possible to calculate the True Positive (TP) and False Positive (FP) (normalized by the object size $\sum GT_f$). For a given frame $f$ and a different OME algorithm $i$, where & is the bitwise AND between matrix, it can calculated by Equation (15):

$$\begin{cases} TP_{fi} = \frac{GT_f \ \& \ OME_{fi}}{\sum GT_f} \\ FP_{fi} = \frac{\overline{GT_f} \ \& \ OME_{fi}}{\sum GT_f} \end{cases}. \tag{15}$$

In Figure 12, it is possible to observe the TP for each frame that is under analyses. The top-5 performing algorithms were picked for a better visualization. The OFC and the Agglomerative algorithms without dimension reduction were the ones that achieved the higher results. Moreover, on Figure 13 is depicted the processing time in ms of the algorithms per frame. The application of dimension reduction does not reduce processing time, according to this Figure 13. This is most likely due to the low complexity of the applied clustering algorithms. The OFC appears as a trade-off between accuracy and processing time.

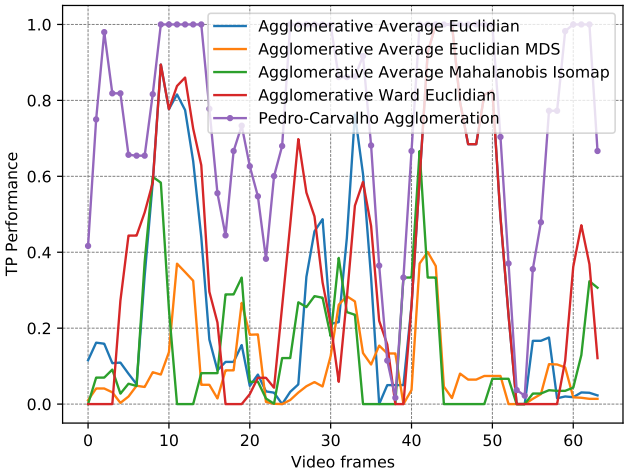



**Figure 12.** Graph analyses of the TP results in the selected video frames.

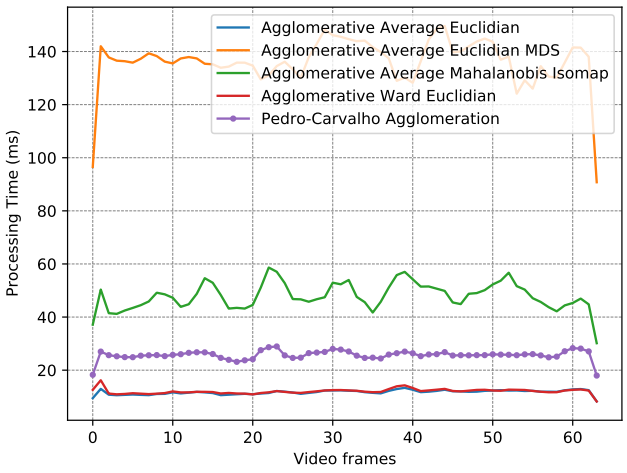



**Figure 13.** Graph analyses of the processing time in the selected video frames.

Just analyzing the TP might be misleading because an algorithm might be detecting bigger regions, which always encapsulate the incoming object and, therefore, outperform other algorithms. On the other hand, when analyzing the FP, some algorithms detect a bigger area around the object, which intuitively is justified by the movement of the object which generate a flow vector between the previous and the current frame. This algorithm should not penalize the algorithm because, ultimately, it is estimating the incoming object correctly. For this reason, a new metric entitled $FPPerformance$ is introduced. The $FPPerformance$ takes into consideration the impact of the error in the decision. For this, after computing the $\overline{GT_f}$ & $OME_{fi}$, for each point of the resulting mask, it has calculated the distance to the nearest point on the object. This new matrix of distances is named as $FD$. Afterwards, the $FP_{Performance}$ can be calculated by:

$$FP_{Performance} = \frac{\sum FD_{fi}}{\sum GT_f \,\&\, OME_{fi}}. \tag{16}$$

Using Equation (16), it is possible to plot the results in Figure 14, which illustrate the FP Performance on the multiple frames. The OFC generate very few points outside the region of the incoming object; therefore, it is always below 0.15 FP Performance, which is a desirable threshold. Values above this threshold might lead to agglomeration of other objects and, therefore, a miss perception of another direction, which could lead to a wrong escape trajectory.

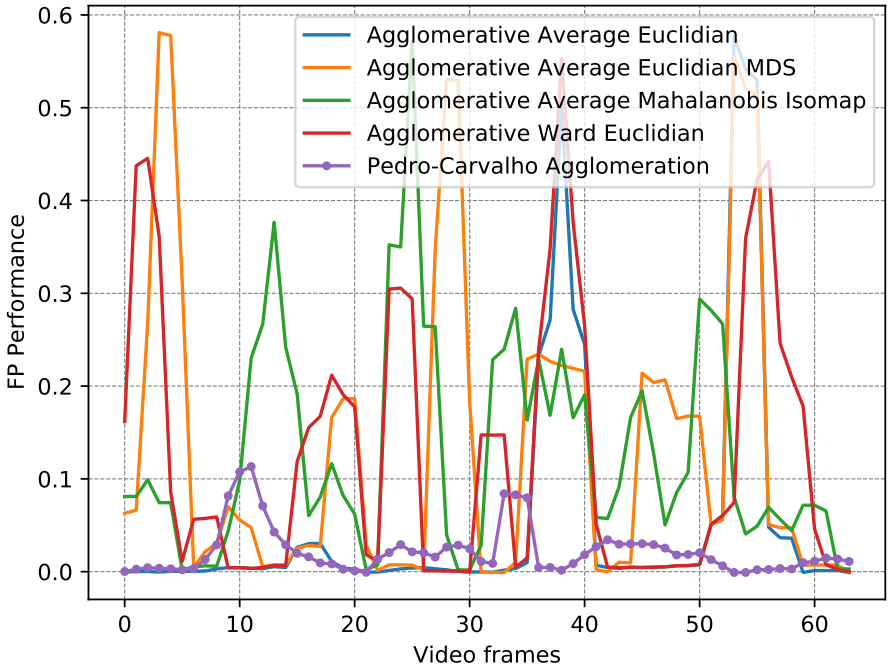



**Figure 14.** Graph analyses of the FP Performance in the selected video frames.

All the results have been summarized in Table 2, which is ordered by decreasing mean FP performance. The TP, FP, and the processing time are presented as the mean of all frames. Furthermore, the Root Mean Square Error (RMSE), and the min/max FP performance results are also presented. It's possible to conclude that for the object trajectory estimation the proposed solution is capable to solve the problem with approximately 2% error. When the algorithm is running live, with continuous frames being fed to the algorithm, the error should statistically decrease because it was measured per frame [64,65]. Further studies are required to analyze the performance impact with the speed variability of the incoming object, the speed variability of the observer, especially angular movements that might induce false trajectories [66]. In addition, environments with multiple moving objects require some attention, as might be the case in most crowded cities.

**Table 2.** Results comparison of the OME algorithm in the selected and annotated 64 frames.

| Algorithm | Distance Metric | Dimension Reduction | Mean TP | Mean FP | RMSE | FP Perf. (Min / Mean / Max) | Mean Time (ms) |
|---|---|---|---|---|---|---|---|
| OFC Agg. | - - | - - | 0.76 | 3.40 | 0.50 | 0.00/0.02/0.06 | 25.99 |
| AA [56] | Euclidian | - - | 0.33 | 0.96 | 0.43 | 0.00/0.06/0.22 | 11.84 |
| AA [56] | Manhattan | - - | 0.33 | 0.96 | 0.43 | 0.00/0.06/0.22 | 8.69 |
| AA [56] | Mahalanobis | - - | 0.33 | 1.29 | 0.46 | 0.00/0.10/0.37 | 33.67 |
| AW [55] | Euclidian | - - | 0.37 | 1.82 | 0.47 | 0.00/0.13/0.27 | 12.23 |
| AW [55] | Manhattan | - - | 0.37 | 1.82 | 0.47 | 0.00/0.13/0.27 | 9.11 |
| AA [56] | Mahalanobis | Isomap [47] | 0.14 | 2.03 | 0.41 | 0.05/0.13/0.24 | 48.51 |
| AA [56] | Euclidian | MDS [48] | 0.10 | 1.43 | 0.37 | 0.02/0.14/0.28 | 137.63 |
| AA [56] | Manhattan | MDS [48] | 0.13 | 1.36 | 0.39 | 0.01/0.14/0.37 | 138.84 |
| Kmeans [53] | Manhattan | - - | 0.43 | 2.43 | 0.48 | 0.01/0.15/0.35 | 25.01 |
| AA [56] | Euclidian | Isomap [47] | 0.09 | 2.18 | 0.48 | 0.06/0.16/0.24 | 26.11 |
| AA [56] | Manhattan | Isomap [47] | 0.09 | 2.18 | 0.48 | 0.06/0.16/0.24 | 21.95 |
| Kmeans [53] | Euclidian | - - | 0.41 | 2.58 | 0.48 | 0.01/0.16/0.45 | 29.10 |
| AA [56] | Mahalanobis | MDS [48] | 0.13 | 1.81 | 0.39 | 0.02/0.17/0.48 | 159.93 |
| AW [55] | Mahalanobis | - - | 0.33 | 2.49 | 0.46 | 0.00/0.18/0.48 | 34.07 |
| Kmeans [53] | Mahalanobis | - - | 0.38 | 3.08 | 0.47 | 0.01/0.19/0.50 | 49.76 |
| Kmeans [53] | Mahalanobis | Isomap [47] | 0.18 | 4.67 | 0.44 | 0.07/0.23/0.41 | 64.07 |
| AW [55] | Mahalanobis | Isomap [47] | 0.18 | 3.91 | 0.46 | 0.07/0.24/0.40 | 48.91 |
| AA [56] | Mahalanobis | t-SNE [51] | 0.16 | 2.92 | 0.42 | 0.06/0.24/0.37 | 1057.79 |
| AW [55] | Mahalanobis | t-SNE [51] | 0.17 | 3.15 | 0.43 | 0.06/0.25/0.39 | 1058.15 |
| Kmeans [53] | Mahalanobis | t-SNE [51] | 0.19 | 3.51 | 0.43 | 0.06/0.26/0.36 | 1075.77 |
| AA [56] | Euclidian | t-SNE [51] | 0.13 | 2.97 | 0.45 | 0.12/0.29/0.43 | 1017.39 |
| AA [56] | Manhattan | t-SNE [51] | 0.13 | 2.97 | 0.45 | 0.12/0.29/0.43 | 1039.02 |
| Kmeans [53] | Euclidian | Isomap [47] | 0.10 | 4.37 | 0.49 | 0.07/0.29/0.44 | 41.11 |
| Kmeans [53] | Euclidian | t-SNE [51] | 0.13 | 3.43 | 0.45 | 0.10/0.30/0.47 | 1034.63 |
| Kmeans [53] | Manhattan | Isomap [47] | 0.08 | 4.70 | 0.47 | 0.07/0.31/0.49 | 36.56 |
| Kmeans [53] | Manhattan | t-SNE [51] | 0.13 | 3.43 | 0.46 | 0.10/0.31/0.47 | 1056.44 |
| AW [55] | Euclidian | t-SNE [51] | 0.11 | 3.31 | 0.47 | 0.09/0.32/0.50 | 1017.83 |
| AW [55] | Manhattan | t-SNE [51] | 0.11 | 3.31 | 0.47 | 0.09/0.32/0.50 | 1039.30 |
| AW [55] | Euclidian | Isomap [47] | 0.11 | 4.97 | 0.49 | 0.08/0.33/0.52 | 26.52 |
| AW [55] | Manhattan | Isomap [47] | 0.11 | 4.97 | 0.49 | 0.08/0.33/0.52 | 22.40 |
| AW [55] | Manhattan | MDS [48] | 0.06 | 4.15 | 0.44 | 0.20/0.42/0.63 | 139.31 |
| AW [55] | Euclidian | MDS [48] | 0.07 | 4.77 | 0.45 | 0.21/0.45/0.90 | 138.09 |
| AW [55] | Mahalanobis | MDS [48] | 0.08 | 4.92 | 0.50 | 0.17/0.52/0.99 | 160.31 |
| Kmeans [53] | Manhattan | MDS [48] | 0.03 | 6.08 | 0.42 | 0.36/0.54/0.78 | 158.46 |
| Kmeans [53] | Mahalanobis | MDS [48] | 0.04 | 5.84 | 0.44 | 0.24/0.54/0.82 | 179.07 |
| Kmeans [53] | Euclidian | MDS [48] | 0.03 | 6.33 | 0.42 | 0.30/0.58/1.00 | 158.76 |

## 5. Conclusions and Future Work

In this work, a safer architecture for UAVs' navigation was presented. This architecture features a block that is responsible for the collision avoidance with dynamic objects (such as a thrown ball). This block uses a NNP composed of a CNN for feature extraction, a RNN for temporal analyses, and a FNN for outputting the detection. Whenever this NNP outputs an incoming collision, an escape vector is calculated by a OME algorithm running in parallel. The OME uses the optical flow between the previous and the current frame, and a clustering algorithm to estimate the trajectory of the incomming object. A novel OF clustering algorithm for this use-case was introduced, which was named OFC, that outperform the state-of-art techniques.

To train the NNP a new dataset with 600 videos, with subjects throwing balls at a UAV was created. The videos were annotated and convert into 37,655 images. The NNP demonstrated an on-time detection, which allows the UAV to estimate a trajectory and to dodged the incoming ball. Both the results on the NNP and the OME demonstrated promising results, achieving 9% error on frame collision detection (multiple consecutive frames drop this percentage) by the NNP, and approximately 2% error on the trajectory estimation by the OME. The tackled use case is just an introduction to the capabilities of

the proposed technique, as it only a scenario that consisted of a thrown ball, and presented a dataset for that purpose. The NNP knowledge can also be transferred to other scenarios with the enlargement of the dataset, and this solution only requires a simple monocular camera, which can be found in most commercial UAVs. The benefits of using these cameras are their small size, less weight, lower power consumption, flexibility, and mounting simplicity. On the other hand, they are highly dependent on weather conditions, might lack image clarity depending on the background color contrast. Regarding the proposed algorithms, the drawbacks identified are the processing requirements of the NNP, which are still not available in most of the out-of-shelf UAVs, and on the professional categories, it is still a significant amount of computational processing and power consumption. In addition, the OME might be accurate in processing the object trajectory, but any minor error can compromise the dodging maneuver. Finally, fast reactions might be dangerous if flying in cluttered environments, or if the UAV has considerable dimensions.

Compared with the current state-of-art, the proposed approach can be applied to standard UAVs using regular video sensors. Even though, in this paper, only the collision of an incoming thrown ball was explored, the authors believe that the algorithm can be easily adapted to multiple use-cases (with static or dynamic obstacles) by increasing the dataset scenarios. The solutions in the literature are in comparison harder to apply or unable of handling fast-moving objects.

This work will be continued with further updates on the modules presented in this article. The list below summarizes some of the key innovative ideas that will drive the future work:

- NNP improvement to estimate an escape vector, as postulated in the ColANet dataset;
- Optical Flow with depth estimation (using an depth camera), allowing the estimation of the distance to the object, therefore adjusting the escape speed, and facilitating the selection of the nearest object.
- CUDA implementation of the OFC algorithm to speed up computation time.
- Live tests on HEIFU hexacopters with the algorithms taking advantage of the on board GPU.

**Author Contributions:** All authors, D.P., J.P.M.-C., J.M.F., and A.M. have contributed equally to the work reported. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work is partially supported by funding from the ECSEL Joint Undertaking (JU) under Grant agreement number: 101007273. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Sweden, Netherlands, Germany, Spain, Denmark, Norway, Portugal, Belgium, Slovenia, Czech Republic, Turkey; and partially funded by Fundação para a Ciência e a Tecnologia under Projects UIDB/04111/2020, foRESTER PCIF/SSI/0102/2017, and IF/00325/2015. This project has also received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 783221. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Belgium, Czech Republic, Finland, Germany, Greece, Italy, Latvia, Norway, Poland, Portugal, Spain, Sweden.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Acknowledgments:** The authors would like to thank the R&D Team of PDMFC and Beyond Vision. Furthermore, the authors like to express their gratitude to Didier Lopes review.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Patias, P. Introduction to Unmanned Aircraft Systems. *Photogramm. Eng. Remote Sens.* **2016**. [CrossRef]
2. Alvarez-Vanhard, E.; Houet, T.; Mony, C.; Lecoq, L.; Corpetti, T. Can UAVs fill the gap between in situ surveys and satellites for habitat mapping? *Remote Sens. Environ.* **2020**. [CrossRef]

3. Navarro, A.; Young, M.; Allan, B.; Carnell, P.; Macreadie, P.; Ierodiaconou, D. The application of Unmanned Aerial Vehicles (UAVs) to estimate above-ground biomass of mangrove ecosystems. *Remote Sens. Environ.* **2020**. [CrossRef]
4. Rödel, C.; Stadler, S.; Meschtscherjakov, A.; Tscheligi, M. Towards autonomous cars: The effect of autonomy levels on Acceptance and User Experience. In Proceedings of the AutomotiveUI 2014—6th International Conference on Automotive User Interfaces and Interactive Vehicular Applications, Seattle, WA, USA, 17–19 September 2014. [CrossRef]
5. Yasin, J.N.; Mohamed, S.A.S.; Haghbayan, M.H.; Heikkonen, J.; Tenhunen, H.; Plosila, J. Unmanned Aerial Vehicles (UAVs): Collision Avoidance Systems and Approaches. *IEEE Access* **2020**, *8*, 105139–105155. [CrossRef]
6. Caron, C. After Drone Hits Plane in Canada, New Fears About Air Safety. 2017. Available online: https://www.nytimes.com/2017/10/17/world/canada/canada-drone-plane.html (accessed on 19 May 2019).
7. BBC. Drone' Hits British Airways Plane Approaching Heathrow Airport. 2016. Available online: https://www.bbc.com/news/uk-36067591 (accessed on 19 May 2019).
8. Canada, C. Drone that Struck Plane Near Quebec City Airport was Breaking the Rules. 2017. Available online: http://www.cbc.ca/news/canada/montreal/garneau-airport-drone-quebec-1.4355792 (accessed on 19 May 2019).
9. BBC. Drone Collides with Commercial Aeroplane in Canada. 2017. Available online: https://www.bbc.com/news/technology-41635518 (accessed on 19 May 2019).
10. Goglia, J. NTSB Finds Drone Pilot At Fault For Midair Collision with Army Helicopter. 2017. Available online: https://www.forbes.com/sites/johngoglia/2017/12/14/ntsb-finds-drone-pilot-at-fault-for-midair-collision-with-army-helicopter/ (accessed on 19 May 2019).
11. Lin, C.E.; Shao, P.C. Failure analysis for an unmanned aerial vehicle using safe path planning. *J. Aerosp. Inf. Syst.* **2020**. [CrossRef]
12. Tellman, J.; News, T.V. First-Ever Recorded Drone-Hot Air Balloon Collision Prompts Safety Conversation. 2018. Available online: https://www.postregister.com/news/local/first-ever-recorded-drone-hot-air-balloon-collision-prompts-safety/article_7cc41c24-6025-5aa6-b6dd-6d1ea5e85961.html (accessed on 19 May 2019).
13. Shakhatreh, H.; Sawalmeh, A.H.; Al-Fuqaha, A.; Dou, Z.; Almaita, E.; Khalil, I.; Othman, N.S.; Khreishah, A.; Guizani, M. Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges. *IEEE Access* **2019**. [CrossRef]
14. Weibel, R.E.; Hansman, R.J. *Safety Considerations for Operation of Unmanned Aerial Vehicles in the National Airspace System*; MIT Libraries: Cambridge, MA, USA, 2005.
15. Zhong, Y.; Hu, X.; Luo, C.; Wang, X.; Zhao, J.; Zhang, L. WHU-Hi: UAV-borne hyperspdectral with high spatial resolution (H2) benchmark datasets and classifier for precise crop identification based on deep convolutional neural network with CRF. *Remote Sens. Environ.* **2020**. [CrossRef]
16. Meinen, B.U.; Robinson, D.T. Mapping erosion and deposition in an agricultural landscape: Optimization of UAV image acquisition schemes for SfM-MVS. *Remote Sens. Environ.* **2020**. [CrossRef]
17. Bhardwaj, A.; Sam, L.; Akanksha.; Martín-Torres, F.J.; Kumar, R. UAVs as remote sensing platform in glaciology: Present applications and future prospects. *Remote Sens. Environ.* **2016**, *175*, 196–204. [CrossRef]
18. Yao, H.; Qin, R.; Chen, X. Unmanned Aerial Vehicle for Remote Sensing Applications—A Review. *Remote Sens.* **2019**, *11*, 1443. [CrossRef]
19. Gerhards, M.; Schlerf, M.; Mallick, K.; Udelhoven, T. Challenges and Future Perspectives of Multi-/Hyperspectral Thermal Infrared Remote Sensing for Crop Water-Stress Detection: A Review. *Remote Sens.* **2019**, *11*, 1240. [CrossRef]
20. Messina, G.; Modica, G. Applications of UAV thermal imagery in precision agriculture: State of the art and future research outlook. *Remote Sens.* **2020**, *12*, 1491. [CrossRef]
21. Gaffey, C.; Bhardwaj, A. Applications of unmanned aerial vehicles in cryosphere: Latest advances and prospects. *Remote Sens.* **2020**, *12*, 948. [CrossRef]
22. Pedro, D.; Matos-Carvalho, J.P.; Azevedo, F.; Sacoto-Martins, R.; Bernardo, L.; Campos, L.; Fonseca, J.M.; Mora, A. FFAU—Framework for Fully Autonomous UAVs. *Remote Sens.* **2020**, *12*, 3533. [CrossRef]
23. Gallup, D.; Frahm, J.M.; Mordohai, P.; Pollefeys, M. Variable baseline/resolution stereo. In Proceedings of the 26th IEEE Conference on Computer Vision and Pattern Recognition, Anchorage, AK, USA, 24–26 June 2008. [CrossRef]
24. Mueggler, E.; Forster, C.; Baumli, N.; Gallego, G.; Scaramuzza, D. Lifetime estimation of events from Dynamic Vision Sensors. In Proceedings of the IEEE International Conference on Robotics and Automation, Seattle, WA, USA, 26–30 May 2015. [CrossRef]
25. Andrew, A.M. *Multiple View Geometry in Computer Vision*; Cambridge University Press: Cambridge, UK, 2001. [CrossRef]
26. Marchidan, A.; Bakolas, E. Collision avoidance for an unmanned aerial vehicle in the presence of static and moving obstacles. *J. Guid. Control. Dyn.* **2020**. [CrossRef]
27. Fan, T.; Long, P.; Liu, W.; Pan, J. Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios. *Int. J. Robot. Res.* **2020**. [CrossRef]
28. van Dam, G.J.; van Kampen, E. Obstacle avoidance for quadrotors using reinforcement learning and obstacle-airflow interactions. In Proceedings of the AIAA Scitech 2020 Forum, Orlando, FL, USA, 6–10 January 2020. [CrossRef]
29. Sampedro, C.; Rodriguez-Ramos, A.; Gil, I.; Mejias, L.; Campoy, P. Image-Based Visual Servoing Controller for Multirotor Aerial Robots Using Deep Reinforcement Learning. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems, Madrid, Spain, 1–5 October 2018. [CrossRef]
30. Zhuge, C.; Cai, Y.; Tang, Z. A novel dynamic obstacle avoidance algorithm based on Collision time histogram. *Chin. J. Electron.* **2017**. [CrossRef]

31. Poiesi, F.; Cavallaro, A. Detection of fast incoming objects with a moving camera. In Proceedings of the British Machine Vision Conference, London, UK, 4–7 September 2017. [CrossRef]
32. Falanga, D.; Kim, S.; Scaramuzza, D. How Fast Is Too Fast? The Role of Perception Latency in High-Speed Sense and Avoid. *IEEE Robot. Autom. Lett.* **2019**. [CrossRef]
33. Romero, A.M. ROS/Concepts. 2014. Available online: http://wiki.ros.org/ROS/Concepts (accessed on 20 May 2019).
34. Kehoe, B.; Patil, S.; Abbeel, P.; Goldberg, K. A Survey of Research on Cloud Robotics and Automation. *IEEE Trans. Autom. Sci. Eng.* **2015**, *12*, 398–409. [CrossRef]
35. AnisKoubaa. Services. 2010. Available online: http://wiki.ros.org/Services (accessed on 20 May 2019).
36. Vooon. Mavros. 2018. Available online: http://wiki.ros.org/mavros (accessed on 20 May 2019).
37. Falanga, D.; Kleber, K.; Scaramuzza, D. Dynamic obstacle avoidance for quadrotors with event cameras. *Sci. Robot.* **2020**, *5*. [CrossRef]
38. Project, D. MAVLink Developer Guide. 2010. Available online: https://mavlink.io/en/ (accessed on 20 May 2019).
39. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018. [CrossRef]
40. Ma, N.; Zhang, X.; Zheng, H.T.; Sun, J. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018.
41. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv* **2017**, arXiv:1704.0486.
42. Boureau, Y.L.; Ponce, J.; Lecun, Y. A theoretical analysis of feature pooling in visual recognition. In Proceedings of the ICML 2010 27th International Conference on Machine Learning, Haifa, Israel, 21–24 June 2010.
43. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**. [CrossRef]
44. Tran, D.; Bourdev, L.; Fergus, R.; Torresani, L.; Paluri, M. Learning spatiotemporal features with 3D convolutional networks. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015. [CrossRef]
45. Schlögl, S.; Postulka, C.; Bernsteiner, R.; Ploder, C. Artificial intelligence tool penetration in business: Adoption, challenges and fears. In Proceedings of the Communications in Computer and Information Science, Vienna, Austria, 20–21 September 2019. [CrossRef]
46. Zach, C.; Pock, T.; Bischof, H. A Duality Based Approach for Realtime TV-L1 Optical Flow. In Proceedings of the Pattern Recognition, 29th DAGM Symposium, Heidelberg, Germany, 12–14 September 2007.
47. Tenenbaum, J.B.; De Silva, V.; Langford, J.C. A global geometric framework for nonlinear dimensionality reduction. *Science* **2000**. [CrossRef]
48. Kruskal, J.B. Nonmetric multidimensional scaling: A numerical method. *Psychometrika* **1964**. [CrossRef]
49. Kruskal, J.B. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* **1964**. [CrossRef]
50. O'Connell, A.A.; Borg, I.; Groenen, P. Modern Multidimensional Scaling: Theory and Applications. *J. Am. Stat. Assoc.* **1999**. [CrossRef]
51. Van Der Maaten, L.J.P.; Hinton, G.E. Visualizing high-dimensional data using t-sne. *J. Mach. Learn. Res.* **2008**, *9, 2579–2605.*
52. van der Maaten, L. Accelerating t-SNE using tree-based algorithms. *J. Mach. Learn. Res.* **2014**, *15, 3221–3245.*
53. MacQueen, J. Some methods for classification and analysis of multivariate observations. In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Oakland, CA, USA, 21 June–18 July 1965.
54. Wu, L.; Garcia, M.A.; Puig, D.; Sole, A. Voronoi-based space partitioning for coordinated multi-robot exploration. *J. Phys. Agents* **2007**. [CrossRef]
55. Müllner, D. Modern hierarchical, agglomerative clustering algorithms. *arXiv* **2011**, arxiv:1109.2378.
56. Murtagh, F.; Legendre, P. Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion? *J. Classif.* **2014**, *31*, 274–295. [CrossRef]
57. Pedro, D.; Mora, A.; Carvalho, J.; Azevedo, F.; Fonseca, J. ColANet: A UAV Collision Avoidance Dataset. *Technol. Innov. Life Improv.* **2020**. [CrossRef]
58. Shanmugamani, R. *Deep Learning for Computer Vision: Expert Techniques to Train Advanced Neural Networks Using Tensorflow and Keras*; Packt Publishing Ltd.: Birmingham, UK, 2018. [CrossRef]
59. Pan, S.J.; Yang, Q. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* **2010**. [CrossRef]
60. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.* **2015**. [CrossRef]
61. Kingma, D.P.; Ba, J.L. Adam: A method for stochastic optimization. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015.
62. Metz, R. *Daredevil Drones: Startup Skydio has Developed a More Sophisticated Autopilot for Drones. Beyond Obstacle Avoidance, It Lets an Aircraft Orient Itself and Navigate through Busy Areas*; Farlex, Inc.: Huntingdon Valley, PA, USA, 2016.
63. Pedro, D.; Lousa, P.; Ramos, A.; Matos-Carvalho, J.; Azevedo, F.; Campos, L. HEIFU—Hexa Exterior Intelligent Flying Unit . In Proceedings of the DECSoS Workshop at SFECOMP 2021. Unpublished work, 2021.
64. Huang, H.; Dabiri, D.; Gharib, M. On errors of digital particle image velocimetry. *Meas. Sci. Technol.* **1997**. [CrossRef]

65. Kazemi, M.; Ghanbari, M.; Shirmohammadi, S. A review of temporal video error concealment techniques and their suitability for HEVC and VVC. *Multimed. Tools Appl.* **2021**. [CrossRef]
66. Wang, Y.; Lin, S. Error-resilient video coding using multiple description motion compensation. *IEEE Trans. Circ. Syst. Video Technol.* **2002**. [CrossRef]