Article

Potential Obstacle Detection Using RGB to Depth Image Encoder–Decoder Network: Application to Unmanned Aerial Vehicles

Tomasz Hachaj

Institute of Computer Science, Pedagogical University of Krakow, 2 Podchorazych Ave, 30-084 Krakow, Poland; tomekhachaj@o2.pl; Tel.: +48-126-627-845

Abstract: In this work, a new method is proposed that allows the use of a single RGB camera for the real-time detection of objects that could be potential collision sources for Unmanned Aerial Vehicles. For this purpose, a new network with an encoder–decoder architecture has been developed, which allows rapid distance estimation from a single image by performing RGB to depth mapping. Based on a comparison with other existing RGB to depth mapping methods, the proposed network achieved a satisfactory trade-off between complexity and accuracy. With only 6.3 million parameters, it achieved efficiency close to models with more than five times the number of parameters. This allows the proposed network to operate in real time. A special algorithm makes use of the distance predictions made by the network, compensating for measurement inaccuracies. The entire solution has been implemented and tested in practice in an indoor environment using a micro-drone equipped with a front-facing RGB camera. All data and source codes and pretrained network weights are available to download. Thus, one can easily reproduce the results, and the resulting solution can be tested and quickly deployed in practice.

Keywords: encoder–decoder network; depth prediction; RGB to depth mapping; obstacle detection; Unmanned Aerial Vehicles; deep neural network

1. Introduction

Encoder–decoder (E-D) deep neural networks (DNN) are currently the primary tool in many digital image processing applications. Particularly popular in these applications are networks with U-Net architectures [1]. In U-Net networks, there are multiple connections of encoder layers with decoder layers through so-called skip connections. Skip connections are used to transfer image features from different levels of resolution of the encoder pyramid, increasing the approximation capability of the network. These types of architectures find application in the segmentation of plants [2], people’s clothes or hair [3], or, for example, medical images [4]. Another valuable property of E-D networks is the capacity for intermodality mapping and prediction using images of one modality to estimate images of another modality. This applies, for example, to RGB image to eye-tracking-based saliency map prediction [5] or, for example, to RGB to depth image prediction.

In the past few years, there have been a number of interesting papers proposing E-D to predict the distances of objects in the image from the camera lens. The authors most often focus on two issues: increasing the accuracy of the prediction [6–12] and increasing the speed of the network [13–16]. Speeding up network performance is achieved by reducing the computational complexity, usually by reducing the number of parameters. Porting models to mobile devices requires the conversion of the weights to a smaller number of bytes (from 4 bytes to 2 or even 1 byte). Both reducing the number of model weights and reducing their size (bytes) often requires a trade-off with model accuracy. Methods that allow RGB-based depth prediction have many practical applications. They allow one to use a single RGB camera to estimate the distance to objects. This is important in situations where...
when one wants to reduce the number of sensors and energy consumption and thus the cost of an embedded system. This finds its application, for example, in small Unmanned Aerial Vehicles (UAVs, drones). UAVs have shown great potential in many fields—for example, in agriculture [17] or various types of surveillance [18].

It is also possible to train depth estimation methods without using ground truth (depth) data. These methods are called “unsupervised” and can be treated as more generalized versions of depth estimation with depth measurements available for training. Several possible solutions to this problem have been proposed. There is a group of algorithms that use a video stream for this task by processing ego-motion [19–21]. On the other hand, unsupervised single-frame methods most often use images acquired from several cameras simultaneously and estimate disparity maps based on them [22–25]. For example, the network described in the paper [13] is an E-D similar to U-Net-like networks with multiple small decoders working at different resolutions, directly on a pyramid of images or features.

There have been many proposed approaches for obstacle detection and collision avoidance for UAVs that use mainly various object detection methods [26–29] or depth maps measured directly by depth/distance sensors [30,31] or stereo cameras [32]. The work of [33] uses a method that acquires a small set of images while a quadrotor is hovering, from which the authors compute a dense depth map. Due to the acquisition time, however, this method is not a real-time method. An extensive survey on obstacle detection and collision avoidance can be found in [34]. Obstacle detection is a different issue from the mono-ocular and multi-ocular SLAM [35]. SLAM is designed to localize and map the environment, and algorithms of this type eliminate, among other things, moving objects from the final map of the environment that could potentially be a source of collisions. Moreover, modern visual SLAMs use sparse feature maps [33,36], which do not map potential obstacles in the paths of vehicles.

UAVs are increasingly used in precision agriculture, both outdoors (in open fields) and indoors (in greenhouses). The materials presented in the study [37] highlight these applications and provide a comprehensive overview that emphasizes the importance of SLAM for UAV solutions in such applications.

Based on the literature, it can be noted that no work has been published that reports the use of RGB to depth mapping from an RGB camera as a data source for an algorithm that predicts potential UAV collisions in real time. In this work, an encoder–decoder network with real-time performance was proposed to estimate the distance map using only a single RGB image (single frame). This network was then used as part of an algorithm to detect objects that could be a potential source of collision for the UAV. Because of the fact that the proposed method uses a single RGB sensor, it can be used in low-cost, lightweight, and low-power-consumption UAVs equipped only with a video camera. It is not necessary to use LIDAR or any other type of distance sensor to detect a potential obstacle in the drone’s path. This is an important financial advantage over more expensive systems of more complex UAVs.

The entire solution was implemented and tested in practice in an indoor environment using a micro-drone equipped with a front-facing RGB camera. All data and source codes and pretrained network weights are available to download. Thus, one can easily reproduce the results, and the resulting solution can be tested and quickly implemented in practice.

2. Materials and Methods

In this section, the proposed DNN is introduced together with the training and validation datasets. Moreover, an algorithm for potential obstacle detection and its implementation with a UAV is described.

2.1. Depth Estimation Network

The proposed network has been inspired by [6], which is a reliable and relatively fast architecture, albeit with too slow performance to be used in real-time solutions. In
order to increase speed while preserving the efficiency of network performance, I propose an architecture that significantly reduces the number of DNN coefficients from 42.8 M (millions) to 6.3 M. The new proposed network is a U-net-inspired encoder–decoder with a DenseNet169 [38] backbone. DenseNet169 is pretrained on ImageNet [39]. Compared to the model proposed in the paper [6], the size of the E-D pyramid has been reduced from 4 to 2 skip connections. The output from the encoder is the fourth pooling layer of DenseNet169. The network architecture is presented in Figure 1.

![Figure 1. Deep encoder–decoder architecture of proposed network for RGB to depth image prediction.](image)

Let us assume that $a_i$ is a ground truth depth image and $\hat{a}_i$ is an image with predicted depth values. Index $i \in [1, \ldots, n]$.

The utilized loss function, as in [6], is a three-element function using the following components:

- Point-wise depth loss for image index $i$:
  
  $$ l_{\text{depth},i} = \text{mean}(|\hat{a}_i - a_i|), \quad (1) $$

  where $\text{mean}(X)$ is the averaged value of matrix $X$ elements.

- Edge-wise loss for image index $i$:
  
  $$ l_{\text{edges},i} = \text{mean}\left( \left| \frac{\partial \hat{a}_i}{\partial x} - \frac{\partial a_i}{\partial x} \right| + \left| \frac{\partial \hat{a}_i}{\partial y} - \frac{\partial a_i}{\partial y} \right| \right), \quad (2) $$

- Structural similarity (SSIM) index [40] loss for image index $i$:
  
  $$ l_{\text{ssim}} = \text{clip}\left( \frac{1 - \text{ssim}(\hat{a}_i, a_i, \text{maxdepth})}{2}, 0, 1 \right) \quad (3) $$

  where $\text{clip}(x, 0, 1)$ is an element-wise value clipped to the range $(0, 1)$ and $\text{maxdepth}$ is the maximal value of the depth pixel in the image.

The final form of the loss function is:

$$ l_i = w_1 \cdot l_{\text{ssim},i} + w_2 \cdot l_{\text{edges},i} + w_3 \cdot l_{\text{depth},i} \quad (4) $$

where $w_1 = 1$, $w_2 = 1$, $w_3 = 0.1$, as was recommended in [6].

Image augmentation during the training consists of color modification and mirroring.
2.2. Obstacle Detection

RGB to depth mapping networks are unfortunately not very accurate. This is perfectly evident in Figure 2, in which point clouds generated using the network proposed in Section 2.1 are presented. Unfortunately, the lack of stability of the image and the disturbances resulting from the incorrect estimation of the distance do not allow (yet) the use of any of the architectures of this type of network for direct and reliable distance measurement. Instead, networks of this type can estimate the mutual positions and distances of objects relative to the camera. Thanks to this, one can estimate which objects are farther away and which are closer. Based on this property, I propose a new algorithm to detect potential obstacles that are in front of the UAV. It consists of several steps, which include distance estimation using DNN, depth map clustering using the DBSCAN algorithm [41], and then using a series of thresholding with adaptive parameters to extract the object that is in a collision path with the drone, if such an object is present.

![Figure 2. Examples of distance estimation inaccuracies visualized with point clouds. Top row (a–c) contains RGB image; bottom row contains depth estimation (d–f).](image)

The algorithm proposed in this work, Algorithm 1, has the following parameters:

- rgb—image;
- scale—scaling parameter to lower computational complexity;
- \((\epsilon, \mu)\)—DBSCAN parameters (epsilon and minimal number of samples);
- \(\alpha\)—fraction of clusters to be considered foreground, where 1 means that all clusters are considered to be potential obstacles; 0.5 means that 50% of all clusters are taken as potential obstacles; clusters are ordered from the nearest to the camera to the farthest;
- \(\beta\)—maximal averaged distance of cluster elements from the camera, under which the cluster is considered as the foreground (potential obstacle), in range \([0, 256]\), where 256
means that regardless of the averaged cluster distance from the camera, its elements are considered as foreground;

- ROI—region of interest in image that represents the further position of the UAV on its forward trajectory;
- $\eta$—if the fraction of non-zero values in ROI in the potential obstacle detection array is higher than $\eta$ threshold, there is an obstacle on the UAV trajectory;
- dnn—deep neural network for depth image estimation from RGB image.

After testing, the following values were adopted to tune the parameters of the algorithm, which I will use for the rest of the work: $s = 4$, $\epsilon = 5$, $\mu = 10$, $\alpha = 0.5$, $\beta = 96$, $\eta = 0.3$. For an input image with a resolution of $640 \times 480$, the network returns a depth map with a resolution of $320 \times 240$. For $s = 4$, the DBSCAN segmented image has a resolution of $80 \times 60$. For this resolution, a $20 \times 20$ ROI was adopted, with its center point 20 pixels up from the center of the image. The ROI can be seen as a rectangle in Figures 3, 5, and 6. The rectangle is green if no obstacles are detected or red if obstacles are detected. Note that the horizontal stretch of the rectangle is an effect of visualization, in order to ensure that the image shape matches the figures.

2.3. Integration with Unmanned Aerial Vehicle System

For the real-time algorithm testing on the UAV, I have used a Tello drone, which is a popular platform to evaluate deep learning on UAVs due to the convenient API [42–45]. Tello enables real-time video streaming and remote control using the UDP protocol. The drone camera has been calibrated using a pinhole camera model using chessboard calibration patterns [46]. The UAV was an 80 g quadcopter with 3-inch propellers and a 1.1 Ah/3.8 V battery that supplies maximally 13 min of flight. The UAV is controlled over 2.4 GHz 802.11n Wi-Fi using the UDP protocol. The maximal speed is 8 m/s; however, because of inertia between steering commands and drone reaction caused by the communication protocol, I used 50% of the available speed to achieve a reliable reaction. The drone is equipped with a stationary 5 MP camera positioned in the front of the vehicle, which has an 82.6° field of view. The re-projection error, calculated as the absolute norm between undistortion transformation and the corner finding algorithm [46], equals 0.094.

The architecture of the system combining Algorithm 1 with the drone is shown in Figure 3. It consists of a drone communicating using WiFi with a remote machine that runs three threads: RGB processing thread, depth processing with obstacle detection thread (implementation of Algorithm 1), and user interface, which enables drone remote navigation for testing purposes. Source codes for this system are also available to download, together with all other algorithms introduced in this paper.

![Diagram of the architecture of the system combining Algorithm 1 with UAV.](image.png)

Figure 3. Diagram of the architecture of the system combining Algorithm 1 with UAV.
Algorithm 1: Algorithm of potential obstacle detection from RGB image

**Data:** Input parameters: Input: RGB, scale, (\(\epsilon, \mu\)), \(\alpha, \beta\), ROI, \(\eta\), dnn

**Result:** Binary information (TRUE/FALSE), if there is obstacle in front of an UAV

// estimate depth map with dnn
depth ← dnn(RGB);

// downscaling with linear interpolation of depth image by factor scale
depth ← resize(depth, size(depth)/scale);

// initialize a set that will hold data for DBSCAN
X ← \(\emptyset\);
id ← 0;

// loop by all x,y coordinates of depth image
for \(x, y\) in size(depth) do
  // assign points from depth image to a set, each element of the set is a three-dimensional vector
  X[id] ← [scale * x, scale * y, depth[x, y]];
id ← id + 1;
end

// DBSCAN clustering
labels ← DBSCAN(X, \(\epsilon, \mu\));

// list with average distance of cluster elements from camera
meandist ← \(\emptyset\);

// loop by all labels
for \(l\) in labels do
  // take all elements with the same label \(l\) and calculate the mean distance from camera
  meandist[l] ← mean(X where labels = \(l\));
end

// sort the mean distance list by ascending order
meandist ← sort(meandist);

// create array filled with zeros with same dimension as depth array
detection ← zeros(size(depth));

// loop by all x,y coordinates of detection array
for \(x, y\) in size(detection) do
  // get cluster label of point \((x, y)\)
  \(l =\) labels(x,y);
  // if order of label \(l\) is smaller than fraction of clusters to be considered as foreground
  if ord(l, meandist) < \(\alpha \cdot count(meandist)\) then
    // if mean distance of cluster elements is below threshold \(\beta\)
    assign point with coordinates \((x,y)\) as potential obstacle
    if meandist[l] < \(\beta\) then
      detection[x,y] = meandist[l];
    end
  end
end

// if number of nonzero pictures in detection image in ROI divided by ROI size is above \(\eta\), there is obstacle on the UAV course
if nonzero(detection[ROI]) / size(ROI) > \(\eta\) then
  return TRUE
else
  return FALSE
end
2.4. Dataset

The algorithm was tested in an indoor environment. This is because the drone that was used is not suitable for flying outdoors due to its small size. For this reason, the training set that I used consisted of pairs of RGB images and distance maps taken indoors. I used the NYU-Depth V2 dataset [47], which is composed of video sequences from a variety of indoor scenes as recorded by both the RGB and depth cameras. The dataset is available at: https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html, accessed on 12 July 2022. The set has 50,688 training images and 654 test images.

3. Results

The network described in Section 2.1 has been implemented in Python 3.8. Among the most important libraries, Keras 2.8 with Tensorflow 2.8 for deep neural network modelling and calculation, and opencv-python 4.5.5 for general-purpose image processing have been used. I have utilized scikit-learn 1.0.2 for DBSCAN implementation. A pretrained model of VGG16 has been downloaded with the Keras Applications 1.0.8 package. All libraries were installed by PIP. Network training and evaluation was carried out on a PC equipped with an Intel i7-9700 3 GHz, 64 GB RAM, and an NVIDIA GeForce RTX 2060 GPU on Windows 10 OS. The CUDA support for Tensorflow was enabled. For the optimization of network weights, the stochastic gradient descent Adam optimizer [48] was utilized. The learning rate was set to $10^{-4}$ with batch size 2. The implementation is partially based on the source codes of paper [6], available at https://github.com/ialhashim/DenseDepth, accessed on 12 July 2022.

I have used third-party communication libraries to establish connection and video data streams from the drone using the DJITelloPy package https://github.com/damiafuentes/DJITelloPy, accessed on 12 July 2022. The proposed algorithm implementation, including the training algorithms, the evaluation algorithm, the drone application used during validation, and the application to generate video based on the results, can be downloaded from https://github.com/browarsoftware/tello_obstacles, accessed on 12 July 2022.

Network training was performed for 40 epochs and lasted approximately 53 h. Loss and validation loss results are shown in Figure 4.

The performance of the proposed network compared to other architectures that allow for distance (depth) map prediction is shown in Table 1. The metrics used were six metrics that are widely accepted to compare new methods against the state-of-the-art [49]:

- average relative error (lower is better):
  \[
  REL = \frac{1}{n} \sum_{i}^{n} \frac{|\hat{a}_i - a_i|}{a_i},
  \]

- root mean squared error (lower is better):
  \[
  RMS = \sqrt{\frac{1}{n} \sum_{i}^{n} (\hat{a}_i - a_i)^2},
  \]

- average (log10) error (lower is better):
  \[
  \log_{10} = \frac{1}{n} \sum_{i}^{n} |\log_{10}(\hat{a}) - \log_{10}(a)|,
  \]

- threshold accuracy (higher is better):
  \[
  \delta_j = \frac{\#(\max(b, \hat{a}) < t_j)}{n},
  \]

where $t_1 = 1.25$, $t_2 = 1.25^2$, $t_3 = 1.25^3$. 


Figure 4. Loss curves for network introduced in Section 2.1. Training has been done using dataset from Section 2.4.

Table 1. Comparison of performance of various depth estimation neural networks on the NYU-Depth-v2 dataset. The results are reported from the original papers. Last row is the results of the proposed network. The second column shows numbers of parameters in millions (M).

| Method                        | #Params (M) | $\delta_1 \uparrow$ | $\delta_2 \uparrow$ | $\delta_3 \uparrow$ | REL ↓ | RMS ↓ | $\log_{10} \downarrow$ |
|-------------------------------|-------------|----------------------|----------------------|----------------------|-------|-------|------------------------|
| Eigen D. et al. [11]          | -           | 0.769                | 0.950                | 0.988                | 0.158 | 0.641 | -                      |
| AdaBins [9]                   | 78          | 0.903                | 0.984                | 0.997                | 0.103 | 0.364 | 0.044                  |
| Alhashim I. and Wonka P [6]   | 42.8        | 0.846                | 0.974                | 0.994                | 0.123 | 0.465 | 0.053                  |
| Fu H. [12]                    | -           | 0.828                | 0.965                | 0.992                | 0.115 | 0.509 | 0.051                  |
| Huang K. et al. [8]           | 32.4        | 0.859                | 0.972                | 0.993                | 0.122 | 0.459 | 0.051                  |
| X. Tu et al. [16]             | 5.0         | 0.782                | -                    | -                    | -     | -     | -                      |
| Wofk D. et al. [15]           | 3.9         | 0.775                | 0.911                | 0.972                | 0.211 | 0.599 | 0.079                  |
| Kerim Yucel M. et al. [14]    | 2.6         | 0.790                | -                    | -                    | -     | -     | -                      |
| This method                   | 6.3         | 0.819                | 0.965                | 0.992                | 0.139 | 0.587 | 0.059                  |

The relatively small number of parameters of the network proposed in this work allowed it to significantly speed up its performance compared to the [6] architecture. The average processing time of a 640 $\times$ 480 resolution image for [6] is 0.104 (9.6 FPS) seconds and that of the proposed architecture is 0.058 seconds (17.2 FPS), on the hardware architecture whose specifications were given at the beginning of this section. This means that the proposed network has 6.79 times fewer parameters and runs 1.79 times faster. This performance is fast enough to achieve successful and reliable work with a 30 FPS video stream for the tested UAV. According to Table 1, the proposed network tends to have lower efficiency than architectures with more parameters, with the exception of [11], while it has the highest $\delta_1$ among architectures with a small number of parameters. Thus, the proposed architecture has a good trade-off between the number of parameters and the efficiency of operation. As can be seen, it is perfectly suitable for real-time system needs.

Based on the observations made during the experiments, the network is able to judge the distances of objects located at a minimum distance of approximately 20 cm from the
camera. If the objects are closer, the network does not work properly recognize the objects’ textures as separate objects.

In order to evaluate the potential obstacle detection algorithm, a set of 112 manually controlled test flights were performed, during which the drone encountered obstacles of various types. The on-board camera images during these flights were analyzed with the proposed Algorithm 1. The algorithm checked whether there were obstacles on the drone’s flight trajectory that threatened to collide with the drone if the drone continued its flight in that direction. Performance was evaluated using an approach similar to a confusion matrix. There were four possible situations: true positive (TP) means correctly detecting an obstacle that the drone could collide with if it continues to fly straight ahead. False positive (FP) means false detection of an obstacle that the drone could collide with if it continues to fly straight ahead. True negative (TN) means that the drone could fly safely. False negative (FN) means no detection of an obstacle in the drone’s path. During the experiment, the drone moved through an indoor space (laboratory room) that was 7.20 m long, around 2 m wide, and around 4 m high. The room was artificially lit, and the windows were covered with blinds. The room contained office furniture such as desks, boxes, chairs, etc. For safety reasons, the drone never moved towards an unshielded person. There were the following obstacles types in the drone’s path:

- Static obstacles of different sizes; see Figure 5a–e. In total, 34 flights were conducted, during which there were obstacles in the drone’s path as well as safe routes.
- Moving (dynamic) obstacles that appeared in front of the drone when it was not moving; see Figure 5f,g. Fifty-four tests on this type of obstacle were recorded. In 22 recordings, the moving obstacle did not cause a collision with the drone. In 32 recordings, the moving obstacle was on a collision track with the drone.
- Moving (dynamic) obstacles that appear in front of the drone—see Figure 5h,i—when the drone is moving forward. Twenty-four tests with this type of obstacle were recorded. In 12 of the recordings, the moving obstacle was in the drone’s path; in the other 12, the drone flew over the obstacle.

The drone was piloted toward these obstacles on both a collision course and a course to avoid them at safe altitudes. The drone also moved towards the walls, doors, and windows of the room. Walls, windows, and doors were also treated as obstacles that the algorithm should warn against. If Algorithm 1 correctly detected all obstacles, the flight was TP. If Algorithm 1 did not detect on the safe path of an obstacle, the flight was TN. If the algorithm made a mistake, it was FP or FN, respectively. A single mistake determined the rating of the algorithm in the corresponding flight as wrong (F). This means that, for each flight, the proposed algorithm receives one rating, correct (P) or wrong (F), indicating whether the algorithm detected all potential obstacles during it, as well as whether it made an FP error. Both erroneous and correct defections lasting less than 0.5 s were disregarded. This assumption was made because of observable artefacts in the camera images resulting from image noise. Evaluation of the results was done manually. All visualizations of the algorithm’s performance in the form of video recordings are available at https://github.com/browarsoftware/tello_obstacles, accessed on 12 July 2022. Sub-images in Figures 5 and 6 are single frames of animation from these recordings. Drone camera data in the form of *.png files are available by contacting the author.
Figure 5. Different types of obstacles used during Algorithm 1 testing. In each sub-image, on the left is an RGB image, at the top right is a depth map estimated by proposed E-D, and at the bottom right are potential obstacles in the drone’s path as detected by Algorithm 1. If the rectangle is red, the algorithm predicts that the drone may collide with the obstacle. If the rectangle is green, the algorithm decides that there are no obstacles in the path. (a) Static obstacle 60 $\times$ 50 $\times$ 75 cm. (b) Static obstacle 60 $\times$ 40 $\times$ 80 cm. (c) Static obstacle with height 120 cm. (d) A second static obstacle with height 120 cm. (e) Static obstacle with a 45 cm deep “valley”. (f) Dynamic obstacle that is not on the drone’s flight trajectory. (g) A dynamic obstacle on a drone’s flight trajectory. (h) Dynamic obstacle that is not on the drone’s flight trajectory. (i) Dynamic obstacle on the drone’s flight trajectory.

Overall, 16 of the 112 tests showed an Algorithm 1 error. One error was of type FP, when the height of a dynamic obstacle was incorrectly estimated; see Figure 6a. The remaining 15 errors were of type FN. Four times, the algorithm failed to detect a potential collision with a window—see Figure 6b—and once with a wall—see Figure 6c—and, once, it misjudged the structure of a two-part complex obstacle—see Figure 6d. The algorithm failed seven times to identify the appearance in front of a hovering drone of a dynamic obstacle with which it could potentially collide; see Figure 6e. Twice, it failed to identify the dynamic obstacle during flight with which it could potentially collide; see Figure 6f.
Figure 6. Example errors of Algorithm 1. (a) Misestimation of obstacle height. (b) Potential collision with a window. (c) Potential collision with a wall. (d) Misjudging the size of a complex obstacle. (e) Misjudging the distance to a dynamic obstacle from a hovering drone. (f) Misjudging the distance to a dynamic obstacle from a moving drone.

4. Discussion

According to the results presented in Section 3, Algorithm 1 proved to be an effective solution for the real-time detection of both stationary and moving obstacles. The loss function graph shown in Figure 4 shows that the training of the network proposed in Section 2.1 runs stably. The algorithm makes virtually no FP errors, as an error of this type appeared in <1% of all tested cases. Most of the errors made by the proposed algorithm were due to misjudgement of distance by the neural network. Note that the training set did not contain objects that would be placed in the air, as in Figure 6e, so the network may not have learned to recognize them correctly. Moreover, when the camera was too close to an object, the algorithm did not work properly (see Figure 6f). Such errors can be difficult to eliminate, because single-frame mono-ocular depth estimation cannot judge the scale of objects and can, with proper camera positioning, interpret, for example, a set of boxes lying on a table as a furnished room. Based on the observations made during the experiment, the minimum effective operating distance of the network is approximately 20 cm from an object. At smaller distances, the network gives erroneous results by not recognizing, for example, the surfaces of walls or doors. The training set also does not include detailed images of windows and window sills. For indoor solutions, they should be included in the training dataset.

The most important limitation of Algorithm 1 is that it is based on a network that does not accurately count distances, but only estimates them, allowing at most the determination of which objects are closer and which are further away. It is therefore necessary to use heuristics that include the parameters of the DBSCAN algorithm and the values of $\alpha$, $\beta$, and $\eta$, which may vary slightly for different distance estimation networks and camera focal lengths. The proposed distance estimation algorithm was also tested on a Logitech HD 1080p webcam, and for identical algorithm parameters, the results for the distance estimation and segmentation of nearby objects were visually almost identical to those of the drone camera. Thus, it can be assumed that, for a network with sufficient performance as measured by (5)–(8), the selection of a depth-estimating E-D is not critical to the performance of the algorithm. By “sufficient”, I mean with measures (5)–(8) not worse than the network proposed in this paper. It is also possible that, if there are many objects at a short distance from an obstacle, e.g., <1.5 m, the DBSCAN algorithm will segment these objects and the threshold alpha will skip obstacles that are at a greater distance than 1.5 m.
in the drone’s path. However, this case did not occur in practice in the experiment. As was mentioned, the most important limitation of Algorithm 1 is that it is based on a network that does not accurately count distances, but only estimates them. Unfortunately, not only Algorithm 1 but all up-to-date E-D networks for single-frame depth estimation cannot be used as reliable sources of distance measurements for SLAM algorithms. All single-frame E-D networks generate similar inaccuracies to those that are presented in Figure 2.

The UAV on which Algorithm 1 was tested is designed to fly in indoor environments; however, the room in which I performed the tests was large enough that its walls were not a significant obstacle. The usefulness of Algorithm 1 in outdoor environments should be tested on another UAV; however, in my opinion, there is no contraindication for Algorithm 1 to be used also in outdoor environments.

5. Conclusions

The algorithm for potential obstacle detection using only an RGB camera applied to Unmanned Aerial Vehicles presented in this work has proven to be an effective and efficient method. To the best of my knowledge, this is the first proposed use of a deep encoder–decoder neural network in an application that allows small drones equipped only with an RGB camera to estimate a dense distance map and detect potential obstacles in real time. The algorithm evaluation results presented in Sections 3 and 4 prove that the proposed method is reliable in the task of detecting potential obstacles. With open-source code available for download and a ready-made Python project structure, the method can be the first choice for both researchers and industry. All these facts might make the proposed approach the preferred algorithm for use in small, low-power-consuming UAV systems with a limited number of sensors. There are several subjects worth further research. First, it would be beneficial to optimize the algorithm for energy savings so that it can be used in an embedded UAV system. In this case, it may require the use of a slightly larger drone than the one used in the experiment described in this work. The second favorable eventuality is to transfer the computation to a portable system based on microcomputers, which are now perfectly capable of handling complex image processing [50]. For E-R RGB to depth mapping networks, a TPU co-processor such as Edge [51] may be required to achieve the appropriate operating frequency. The bottleneck is DBSCAN, which executes relatively slowly relative to the other elements of Algorithm 1. It would be necessary to test a number of available modifications of this algorithm for their effectiveness [52–54] as an alternative to the “classic” DBSCAN.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Source codes can be downloaded from: https://github.com/browarsoftware/tello_obstacles accessed on 12 July 2012.

Acknowledgments: The author would like to thank Łukasz Tomkiewicz for his help during the experiments.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Ronneberger, O.; Fischer, P.; Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Medical Image Computing and Computer-Assisted Intervention—MICCAI 2015, Proceedings of the 18th International Conference, Munich, Germany, 5–9 October 2015; Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 234–241.

2. Wang, J.; Li, B.; Zhou, Y.; Meng, Q.; Rende, S.F.; Rocco, E. Real-time and Embedded Compact Deep Neural Networks for Seagrass Monitoring. In Proceedings of the 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Toronto, ON, Canada, 11–14 October 2020; pp. 3570–3575. [CrossRef]
3. Levinstein, A.; Chang, C.; Phung, E.; Kezele, I.; Guo, W.; Aarabi, P. Real-Time Deep Hair Matting on Mobile Devices. In Proceedings of the 2018 15th Conference on Computer and Robot Vision (CRV), Toronto, ON, Canada, 8–10 May 2018; pp. 1–7. [CrossRef]

4. Yao, Z.; He, K.; Zhou, H.; Zhang, Z.; Zhu, G.; Xing, C.; Zhang, J.; Zhang, Z.; Shao, B.; Tao, Y.; et al. Eye3DVas: Three-dimensional reconstruction of retinal vascular structures by integrating fundus image features. In Proceedings of the Frontiers in Optics/Laser Science, Washington, DC, USA, 14–17 September 2020; Optica Publishing Group: Washington, DC, USA, 2020; p. JTuB1.22.

5. Hachaj, T.; Stolińska, A.; Andrzejewska, M.; Czerski, P. Deep Convolutional Symmetric Encoder-Decoder Neural Networks to Predict Students’ Visual Attention. Symmetry 2021, 13, 2246. [CrossRef]

6. Alhashim, I.; Wonka, P. High Quality Monocular Depth Estimation via Transfer Learning. arXiv 2018, arXiv:1812.11941.

7. Li, X.; Hou, Y.; Wu, Q.; Wang, P.; Li, W. DVOnet: Unsupervised Monocular Depth Estimation and Visual Odometry. In Proceedings of the 2019 IEEE Visual Communications and Image Processing (VCIP), Sydney, Australia, 1–4 December 2019; pp. 1–4. [CrossRef]

8. Huang, K.; Qu, X.; Chen, S.; Chen, Z.; Zhang, W.; Qi, H.; Zhao, F. Superb Monocular Depth Estimation Based on Transfer Learning and Surface Normal Guidance. Sensors 2020, 20, 4856. [CrossRef]

9. Bhat, S.F.; Alhashim, I.; Wonka, P. AdaBins: Depth Estimation Using Adaptive Bins. In Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Virtual, 19–25 June 2021; IEEE Computer Society: Los Alamitos, CA, USA, 2021; pp. 4008–4017. [CrossRef]

10. Ranftl, R.; Lasinger, K.; Hafner, D.; Schindler, K.; Koltun, V. Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-Shot Cross-Dataset Transfer. IEEE Trans. Pattern Anal. Mach. Intell. 2022, 44, 1623–1637. [CrossRef] [PubMed]

11. Eigen, D.; Fergus, R. Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-scale Convolutional Architecture. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015; pp. 2650–2658. [CrossRef]

12. Fu, H.; Gong, M.; Wang, C.; Batmanghelich, K.; Tao, D. Deep Ordinal Regression Network for Monocular Depth Estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018.

13. Poggi, M.; Aleotti, F.; Tosi, F.; Mattoccia, S. Towards Real-Time Unsupervised Monocular Depth Estimation on CPU. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 5848–5854. [CrossRef]

14. Yucel, M.K.; Dimaridou, V.; Drosou, A.; Saà-Garriga, A. Real-time Monocular Depth Estimation with Sparse Supervision on Mobile. In Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Nashville, TN, USA, 19–25 June 2021; pp. 2428–2437.

15. Wofk, D.; Ma, F.L.; Yang, T.J.; Karaman, S.; Sze, V. FastDepth: Fast Monocular Depth Estimation on Embedded Systems. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019.

16. Tu, X.; Xu, C.; Liu, S.; Li, R.; Xie, G.; Huang, J.; Yang, L.T. Efficient Monocular Depth Estimation for Edge Devices in Internet of Things. IEEE Trans. Ind. Inform. 2021, 17, 2821–2832. [CrossRef]

17. Wang, D.; Li, W.; Liu, X.; Li, N.; Zhang, C. UAV environmental perception and autonomous obstacle avoidance: A deep learning and depth camera combined solution. Comput. Electron. Agric. 2020, 175, 105523. [CrossRef]

18. Tullu, A.; Hassanalian, M.; Hwang, H.Y. Design and Implementation of Sensor Platform for UAV-Based Target Tracking and Obstacle Avoidance. Drones 2022, 6, 89. [CrossRef]

19. Mahjourian, R.; Wicke, M.; Angelova, A. Unsupervised Learning of Depth and Ego-Motion from Monocular Video Using 3D Geometric Constraints. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 5667–5675. [CrossRef]

20. Godard, C.; Aodha, O.; Brostow, G.J. Unsupervised Monocular Depth Estimation with Left-Right Consistency. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; IEEE Computer Society: Los Alamitos, CA, USA, 2017; pp. 6602–6611. [CrossRef]

21. Ummelhofer, B.; Zhou, H.; Uhrig, J.; Mayer, N.; Ilg, E.; Dosovitskiy, A.; Brox, T. DeMoN: Depth and Motion Network for Learning Monocular Stereo. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 5622–5631.

22. Flynn, J.; Neuelander, I.; Philbin, J.; Snively, N. Deep Stereo: Learning to Predict New Views from the World’s Imagery. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 5515–5524. [CrossRef]

23. Xie, J.; Girshick, R.; Farhadi, A. Deep3D: Fully Automatic 2D-to-3D Video Conversion with Deep Convolutional Neural Networks. In Computer Vision—ECCV 2016, Proceedings of the 14th European Conference, The Netherlands, 11–14 October 2016; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2016; Volume 9908. [CrossRef]

24. Garg, R.; B G, V.K.; Carneiro, G.; Reid, I. Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue. In Computer Vision—ECCV 2016, Proceedings of the 14th European Conference, The Netherlands, 11–14 October 2016; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2016; Volume 9912. [CrossRef]
25. Poggi, M.; Tosi, F.; Matteuccia, S. Learning Monocular Depth Estimation with Unsupervised Trinocular Assumptions. In Proceedings of the 2018 International Conference on 3D Vision (3DV), Verona, Italy, 5–8 September 2018; IEEE Computer Society: Los Alamitos, CA, USA, 2018; pp. 324–333. [CrossRef]

26. Attigui, M. Autonomous Drone Programming Feature Detection, Tracking, and Obstacle Avoidance; Harz University of Applied Sciences: Wernigerode, Germany, 2021. [CrossRef]

27. Priamibod, A.; Ariffin, F.; Nasuha, A.; Winursito, A. Face Tracking for Flying Robot Quadcopter based on Haar Cascade Classifier and PID Controller. J. Phys. Conf. Ser. 2021, 2111, 012046. [CrossRef]

28. Nair, R.; Al-Assadi, T. Vision-Based Obstacle Avoidance for Small Drone using Monocular Camera. IOP Conf. Ser. Mater. Sci. Eng. 2020, 928, 032048. [CrossRef]

29. Sabetghadam, B.; Alcántara, A.; Capitán, J.; Cunha, R.; Ollero, A.; Pascoal, A. Optimal Trajectory Planning for Autonomous Drone Cinematography. In Proceedings of the 2019 Conference on Mobile Robots (ECMR), Prague, Czech Republic, 4–6 September 2019; pp. 1–7. [CrossRef]

30. Carrio, A.; Vemprala, S.; Ripoll, A.; Saripalli, S.; Campoy, P. Drone Detection Using Depth Maps. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 1034–1037. [CrossRef]

31. Ovrén, H.; Forsssén, P.E.; Törnvqvist, D. Why would i want a gyroscope on my RGB-D sensor? In Proceedings of the 2013 IEEE Workshop on Robot Vision (WORV), Clearwater Beach, FL, USA, 16–17 January 2013; pp. 68–75. [CrossRef]

32. McGuire, K.; de Croon, G.; De Wagter, C.; Tuyls, K.; Kappen, H. Efficient Optical Flow and Stereo Vision for Velocity Estimation and Obstacle Avoidance on an Autonomous Pocket Drone. IEEE Robot. Autom. Lett. 2017, 2, 1070–1076. [CrossRef]

33. Alvarez, H.; Paz, L.M.; Sturm, J.; Cremers, D. Collision Avoidance for Quadrotors with a Monocular Camera. In Experimental Robotics, Proceedings of the 14th International Symposium on Experimental Robotics, Marrakech and Essaouira, Morocco, 15–18 June 2014; Hsieh, M.A., Khatib, O., Kumar, V., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 195–209. [CrossRef]

34. Yasin, J.N.; Mohamed, S.A.S.; Haghhiyan, M.H.; Heikkonen, J.; Tenhunen, H.; Plosila, J. Unmanned Aerial Vehicles (UAVs): Collision Avoidance Systems and Approaches. IEEE Access 2020, 8, 105139–105155. [CrossRef]

35. Mazurek, P.; Hachaj, T. SLAM-OR: Simultaneous Localization, Mapping and Object Recognition Using Video Sensors Data in Open Environments from the Sparse Points Cloud. Sensors 2021, 21, 4734. [CrossRef] [PubMed]

36. Singh, A.; Narula, R.; Rashwan, H.; Abdel-nasser, M.; Puig, D.; Nandi, G. Efficient deep learning-based semantic mapping approach using monocular vision for resource-limited mobile robots. Neural Comput. Appl. 2022, 34, 15617–15631. [CrossRef]

37. Aslan, M.F.; Durdu, A.; Sabanci, K.; Plosila, J. A Comprehensive Survey of the Recent Studies with UAV for Precision Agriculture in Open Fields and Greenhouses. Appl. Sci. 2022, 12, 1047. [CrossRef]

38. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely Connected Convolutional Networks. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 2261–2269. [CrossRef]

39. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2009; pp. 248–255. [CrossRef]

40. Wang, Z.; Bovik, A.; Sheikh, H.; Simoncelli, E. Image quality assessment: From error visibility to structural similarity. IEEE Trans. Image Process. 2004, 13, 600–612. [CrossRef]

41. Ester, M.; Kriegel, H.P.; Sander, J.; Xu, X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. KDD 1996, 96, 226–231.

42. Subash, K.V.; Srinu, M.V.; Siddhartha, M.; Harsha, N.S.; Akkala, P. Object Detection using Ryze Tello Drone with Help of Mask-RCNN. In Proceedings of the 2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Bangalore, India, 5–7 March 2020; pp. 484–490. [CrossRef]

43. Budiyanto, A.; Ramadhan, M.I.; Burhanudin, I.; Triharminto, H.H.; Santoso, B. Navigation control of Drone using Hand Gesture based on Complementary Filter Algorithm. J. Phys. Conf. Ser. 2021, 1912, 012034. [CrossRef]

44. Anwar, A.; Raychowdhury, A. Autonomous Navigation via Deep Reinforcement Learning for Resource Constraint Edge Nodes Using Transfer Learning. IEEE Access 2020, 8, 26549–26560. [CrossRef]

45. Gio, N.; Brisco, R.; Vuletic, T. Control of a drone with body gestures. Proc. Des. Soc. 2021, 1, 761–770. [CrossRef]

46. Zhang, Z. A flexible new technique for camera calibration. IEEE Trans. Pattern Anal. Mach. Intell. 2000, 22, 1330–1334. [CrossRef]

47. Nathan S.; Derek Hoiem, P .K.; Fergus, R. Indoor Segmentation and Support Inference from RGBD Images. In Proceedings of the 12th European Conference on Computer Vision, Florence, Italy, 7–13 October 2012. [CrossRef]

48. Eigen, D.; Puhrsch, C.; Fergus, R. Depth Map Prediction from a Single Image using a Multi-Scale Deep Network. In Advances in Neural Information Processing Systems; Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2014; Volume 27.

49. Hachaj, T. A Method for Human Facial Image Annotation on Low Power Consumption Autonomous Devices. Sensors 2020, 20, 2140. [CrossRef] [PubMed]

50. Yazdanbakhsh, A.; Akin, B.; Seshadri, K.K. An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks. arXiv 2021, arXiv:2102.10423.
52. Jang, J.; Jiang, H. DBSCAN++: Towards fast and scalable density clustering. In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 10–15 June 2019; Volume 97, pp. 3019–3029.

53. Pulabaigari, V.; Veluru, S. Rough-DBSCAN: A fast hybrid density based clustering method for large data sets. Pattern Recognit. Lett. 2009, 30, 1477–1488. [CrossRef]

54. Almassri, H.; Dackermann, T.; Haala, N. mDBSCAN: Real Time Superpixel Segmentation by DBSCAN Clustering based on Boundary Term. In Proceedings of the ICPRAM 2019—8th International Conference on Pattern Recognition Applications and Methods, Prague, Czech Republic, 19–21 February 2019; pp. 283–291. [CrossRef]