CNN-based Single Image Obstacle Avoidance on a Quadrotor

Punarjay Chakravarty, Klaas Kelchtermans, Tom Roussel, Stijn Wellens, Tinne Tuytelaars and Luc Van Eycken[1]

Abstract— **This paper demonstrates the use of a single forward facing camera for obstacle avoidance on a quadrotor. We train a CNN for estimating depth from a single image. The depth map is then fed to a behaviour arbitration based control algorithm that steers the quadrotor away from obstacles. We conduct experiments with simulated and real drones in a variety of environments.**

I. INTRODUCTION

There are a large number of inexpensive, light-weight quadrotor UAVs on the market that come with a front-facing camera and are controllable over Wi-Fi using app-enabled smartphones. These UAVs, in the hands of inexperienced pilots are prone to regular crashes because they have little autonomous flying capability of their own. The problem of crashing into obstacles is even more apparent in obstacle-strewn indoor environments, where most people are liable to first experiment with their Christmas presents, given that outdoor flying is tightly regulated in many countries, especially in Europe. Crashes chip and break the plastic propellers and the body of the drone, and repeated replacement quickly turns out to be an expensive proposition.

Now imagine that users are able to download a software update for the app on their phone that allows their drones to avoid obstacles and fly autonomously with the existing hardware on their quadrotors. This is the application that we target in this paper - single image based depth estimation and obstacle avoidance for quadrotor helicopters.

We train a Deep Convolutional Neural Network (CNN) to predict depth maps given RGB images with supervised learning over thousands of training images collected from both online databases and data that we have collected ourselves. We use a control algorithm in a Behaviour Arbitration scheme that accepts the depth map as its input and outputs the angular velocity $\dot{\phi}$ in both yaw and pitch axes to steer the quadrotor away from obstacles and towards a goal location. We conduct experiments in a simulated environment with different types of horizontal and vertical obstacles and in a real-world indoor environment. Thus, our key contribution is the training of a CNN for generation of depth maps from single images, and the demonstration of a control algorithm operating on these depth maps in a Behaviour Arbitration scheme for real-time control of a quadrotor in indoor environments.

The rest of the paper is organized as follows. Section II describes prior work done in this area. Section III describes the theory behind the CNN and Behaviour Arbitration algorithms. Section IV talks about important implementation details to successfully train the network. Section V describes results from simulated and real-world drone flights for obstacle avoidance. We conclude with Section VI.

[1] The authors are with KU Leuven, ESAT-PSI, iMinds, Belgium. firstname.lastname@esat.kuleuven.be

II. RELATED WORK

A. Traditional Sensing for Obstacle Avoidance

Traditionally, obstacle avoidance for robots has been done using either active sensing like LIDAR, structured light, sonar or IR [1], [2] or using passive sensing like stereo vision [3], [4].

Structure in the scene can also be determined from a single moving camera, and [5] uses the camera movement during quadrotor hover to calculate a depth map using dense Structure from Motion (SfM), calculating a path, and then flying through it. However, in such an SfM-based obstacle avoidance scheme, the drone is limited to a hover-map-plan-path-move-hover trajectory, and even SfM methods that can path-plan on the fly [6] are not able to avoid dynamic obstacles which might have moved during mapping or between mapping cycles. Also, untextured walls (as is the case with most indoor environments) are not detected in traditional SfM methods. In contrast, a single image depth based obstacle avoidance scheme such as ours has the ability to deal with plain, white-washed walls and moving obstacles.

Time of flight sensors like LIDAR and structured light sensors can be expensive, both in terms of actual cost, and in terms of weight and power requirements, both at a premium when it comes to small flying platforms. Sonar and IR are only suitable for very short range (within a metre) depth sensing, and only give information about free space within the small cone of the returned signal, which is suitable for short-range obstacle avoidance, but not for longer range path planning or for fast, reactive flying. Stereo results in depth images with more information, that can be used for path planning, but with the small baseline between cameras possible on a drone, it also suffers from limited range problems.

B. Single Image Based Obstacle Avoidance and Navigation

Single images were first used for depth extraction, using supervised learning by [7]. Large amounts of data - images and their corresponding depth maps were used to train a classifier to create depth maps from single images. Hand-crafted convolutional image features were used to encode a feature vector for local image patches. Contextual information was captured by applying the filters at multiple scales and an

MRF was used to probabilistically model the relationship between the depth of neighbouring patches. Similar hand-crafted features were used by [8], who demonstrated their use for controlling a remote controlled car from a single RGB camera mounted on it, and by [9] who used them for outdoor obstacle avoidance on a quadrotor helicopter.

[10] used a monocular camera and sonar sensors to fly their helicopters in indoor environments. Perspective cues were used to detect the flying direction in corridors and staircases. Edge detection and the Hough Transform were used to detect parallel lines in a staircase and converging lines in a corridor. Sonar was used to avoid obstacles.

More recently, with the success of Deep Learning, CNNs have been used for single image depth extraction [11], [12], [13]. Hand-crafted image features are now replaced by the features that the network learns, given large amounts of training data. CNNs are applied at multiple scales in the image, with the output of the CNNs at larger scales used as an additional input to the CNNs at lower scales to achieve depth maps of higher resolution. A CNN-based algorithm that given an input image, outputs one out of three classes - turn left, go straight and turn right has been used to fly a quadrotor through forest trails [14]. They used 3 head-mounted cameras (pointed straight, left and right) on a hiker walking through trails for training the network that output 3 classes - straight, left and right, which were used for controlling the drone. However, this means that the directional control output is also learnt by the network and is tailored to one particular environment. [15] adapted the fully connected layers of [14]'s network (trained in forests in Switzerland) to work better in forests in the United States using Domain Adaptation. In contrast, we use a Deep CNN to create depth maps that are then fed into a Behaviour Arbitration-based control scheme. It outputs a continuous control signal that works on the estimated single image based depth map, and is not environment specific. Training images are acquired indoors using the Microsoft Kinect, but a stereo setup or a time of flight camera could also be used (outdoors) for training. We train a network for indoor environments, but another network could be trained for depth estimation in outdoor environments, with an unchanged control scheme. In our experiments with both hand-crafted features and CNNs, we find the latter a more promising approach, and more suitable for real-time applications. Our trained network is able to give relative depth information up to 5-10 metres from the camera.

III. Method

A. Single Image Depth Extraction

The architecture of our CNN is based on the Global Coarse Scale Network used by [11] (Figure 1). This network has a sequence of convolutional layers alternated with max-pooling and ReLU layers, followed by 2 fully connected layers.

We added a batch normalization layer at the end of each convolutional layer. This helps against overfitting by normalizing batches of data processed in a particular layer to a Gaussian distribution.

For the network loss function, we use a scale invariant error $L(y, y^*)$ in log-space, between the estimated depth image y and the ground truth depth image y^*. The log-loss d_p is calculated pixel-wise, but is normalized by $\frac{1}{P^2}(\sum_p d_p)^2$, which is the squared log error over all pixels p in the image. This is done to make the equation scale invariant, as in [11].

$$L(y, y^*) = \frac{1}{P}\sum_p d_p^2 - \frac{1}{P^2}(\sum_p d_p)^2 \tag{1}$$

where $\frac{1}{P^2}(\sum_p d_p)^2$ is the squared log error for the whole image and $d_p = \log y_p - \log y_p^*$ is the per pixel log error.

This gives us the relative depth of different obstacles around the quadrotor in the image. Even though we don't have the absolute depth, we can plug in the scale-invariant depth map produced by the network into the control algorithm described in the next section to control the quadrotor.

B. Quadrotor Control based on a Behavior Arbitration Scheme

We use the Behaviour Arbitration scheme from [1] for controlling the quadrotor from a depth map. In simulation, we conduct experiments with both a simulated Kinect sensor on a drone, and the output of the single image based depth estimation described in the previous section. In real-world experiments, we only use the single image depth.

The original behaviour for avoiding obstacles [1] relied on a sonar ring or a laser range finder type 2.5D sensor, with a single strip of depth values around the robot. In our case, we convert the depth map to 2.5D by using depth values from its centre in the vertical and horizontal directions, as shown in Figure 2.

These depth values d_i from angles ψ_i around the quadrotor are then input into the avoid behaviour (equation 3) to get an angular velocity that steers it away from obstacles. Similarly, equation 2 steers the drone toward a goal orientation (ψ_{goal}, relative to the current bearing ϕ) by steering away from the current bearing towards the target bearing. The goto and avoid behaviours are combined using a weighted sum of angular velocities output by each (equation 4) as in [1].

$$\dot{\phi} = f_{goto}(\phi) = -\lambda_{goto} sin(\phi - \psi_{goal}) \tag{2}$$

$$\begin{aligned}\dot{\phi} &= f_{avoid}(\phi) \\ &= \lambda_{avoid}\sum_i [(\phi - \psi_i)\exp(-c_{obst} d_i)\exp(-(\phi - \psi_i)^2/2\sigma^2)]\end{aligned} \tag{3}$$

$$\dot{\phi} = \sum_b w_b f_b(\phi) \tag{4}$$

where the gains λ_{avoid} and c_{obst} can be adjusted, along with angular range σ, to be more or less reactive to the distance to walls and obstacles. Similarly, λ_{goto} can be adjusted to make the drone react more quickly in moving towards the goal. We set the angular range to be equal to the FOV of the camera of the drone. These parameters and their

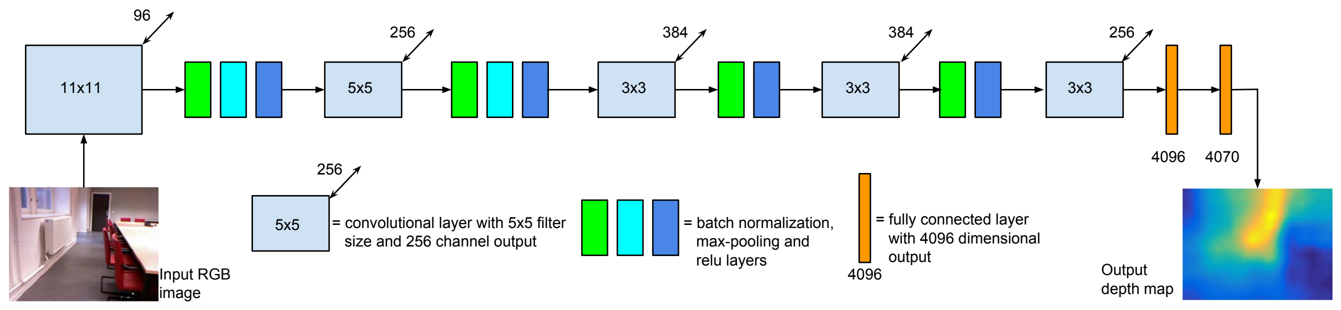
96
11x11
256
5x5
384
3x3
384
3x3
256
3x3
4096
4070
Input RGB image
256
5x5
= convolutional layer with 5x5 filter size and 256 channel output
= batch normalization, max-pooling and relu layers
= fully connected layer with 4096 dimensional output
4096
Output depth map

Fig. 1. CNN architecture for depth estimation from a single image.

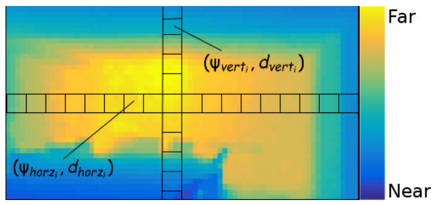

Fig. 2. Extraction of depth values from depth map for Avoid Behaviour. A vertical and horizontal strip through the middle of the depth image is used, with averaged depths within each vertical and horizontal bin $vert_i/horz_i$ used for depth value d_i from angle ψ_i in Equation 3.

effects on obstacle avoidance are explained further in Section V. The avoid behaviour is prone to getting stuck in corners. To solve this, we use a template signature (a triangular or hat function that corresponds to the depth values in a corner) to detect a corner and move away from it if necessary.

IV. IMPLEMENTATION DETAILS

A. Depth CNN

a) Data collection: The depth CNN is initially trained using 260k images from the NYU2 dataset v2 [16]. This dataset contains RGB images and their corresponding depth maps taken in a variety of indoor scenes. The CNN is then fine-tuned with 20k further samples collected from around our offices. These image/depthmap pairs are collected by walking around with a Kinect sensor around the corridors, classrooms and meeting rooms at our university building. We will refer to this dataset as the ESAT dataset in the remainder of the paper.

b) Training: Our network contains about 80 million parameters and we spent a significant amount of time optimizing the hyperparameters - learning rate, momentum, dropout rate and regularization for ridge regression.

A too slow learning rate for Stochastic Gradient Descent causes the network to converge very slowly. A too fast learning rate causes the network to converge quickly, but it converges to a higher loss value. The momentum parameter behaves like the coefficient of friction in the optimization landscape. A schedule that starts with a smaller value (0.5) and increases as the training progresses ensures good convergence. Finally, the regularization parameter for ridge regression needs to be selected so that the network does not overfit. When applied, fine tuning is done by resuming training with new data samples and lower learning rate.

We train our network [1] using the Matconvnet library for Matlab [17]. It takes 12 hours to train on a NVIDIA GTX 1050 GPU.

[1]details available from `http://www.jaychakravarty.com/single-image-obstacle-avoid/`

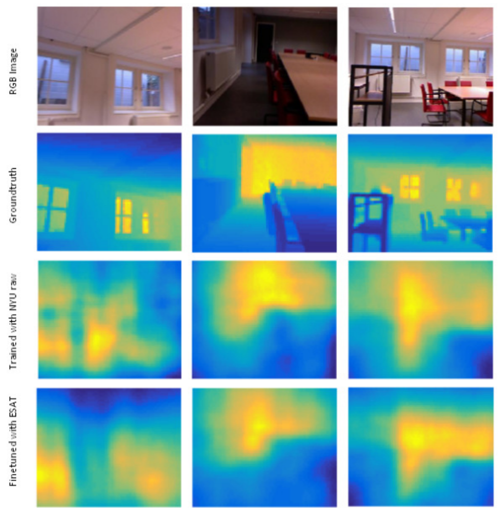

Fig. 3. Depth maps generated by our network on test data from our building (ESAT dataset). From top to bottom: RGB image, ground truth, depth map with network trained on NYU2 dataset, depth map with network fine-tuned on the ESAT dataset.

TABLE I
DEPTH NETWORK COMPARISON

Method	Error (rmse_inv)	Accuracy ($\delta < 1.25$)
Train NYU2/Test NYU2 [11]	0.221	0.618
Train NYU2/Test NYU2	0.181	0.703
Train NYU2/Test ESAT	0.484	0.314
Fine-tune ESAT/Test ESAT	0.332	0.480

B. Controlling the drone

Real-world experiments are performed using a Parrot Bebop drone 2 [18] controlled from a Dell Alienware laptop with 16GB RAM and an NVIDIA 765M GPU that allows the CNN to run at 20 fps. We interface with the drone using ROS, using the bebop autonomy package created and maintained by [19]. The drone sends images to the laptop, which processes each RGB image to create a depth map, and a control output $\dot{\phi}$. We only use the angular velocity in the yaw direction for real-world experiments (both yaw and pitch $\dot{\phi}$ are used in the simulation experiments). The quadrotor is launched at a particular height, and then proceeds forward autonomously at constant forward speed. The single image depth and control algorithm controls its steering.

TABLE II

PERFORMANCE OF OUR ALGORITHMS IN THE 100 SIMULATED WORLDS.

Method	% Completed	Failures
Kinect depth + Control	100%	-
Single Image depth + Control	68%	13 stuck 19 crashes

V. EXPERIMENTS

A. *Performance of the depth network on RGB images*

We test the performance of the depth network on the left out data from the NYU2 and ESAT datasets (480 and 672 test images respectively) using the root mean square invariant (rmse_inv) error and accuracy. The rmse_inv error is calculated over all the test images i using the image-wise scale-invariant error in log-space $L(y_i, y_i^*)$ from equation 1.

$$rmse\_inv = \sqrt{\frac{1}{I}\sum_i L(y_i, y_i^*)} \quad (5)$$

where I is the total number of test images.

Accuracy is determined by

$$\% \text{ of } y_{i,p} \text{ s.t. } max(\frac{y_{i,p}}{y_{i,p}^*}, \frac{y_{i,p}^*}{y_{i,p}}) = \delta < threshold \quad (6)$$

where $y_{i,p}$ and $y_{i,p}^*$ are the estimated and ground-truth depths for pixel p in image i.

Table I shows the performance of the depth network under these criteria under different training and testing regimes. First, using training data from NYU2 and testing on the left out images from NYU2, our network (row 2) is comparable to the coarse network of [11] (row 1), but with slightly improved performance (lower rmse_inv and higher accuracy), which is perhaps attributable to the addition of batch normalization layers. Row 3 shows that our network trained on NYU2 performs worse when tested on ESAT. But when the same network is fine-tuned with 20k images from ESAT, and tested on the left out images from ESAT, its performance improves (row 4). The left out images from ESAT were all taken in rooms not seen in the training set. Qualitative results are shown in Figure 3. These results show that fine-tuning with data from an environment improves the performance of the network in that environment. However, to test the generalization of the original network to different simulated and real-world environments, we conduct all our subsequent experiments with our network trained on the NYU2 dataset.

B. *Performance of the depth network in steering the drone in simulated environments*

We generate 100 simulated worlds in Gazebo to test the flying characteristics of the single image depth and Behaviour Arbitration control discussed earlier. Each world is a rectangular room with 3 types of obstacles: a wall, a ceiling overhang and a bump on the floor. The size and order of the obstacles is varied across the worlds. These obstacles are designed to test the obstacle avoidance properties of the drone in both the yaw and pitch axes. The drone takes off at one end of the room, and is given a target bearing direction. Its current bearing is taken from the ground truth available in the simulation. In reality this would require a compass or localization software based on sensor data. It is expected to navigate the room whilst avoiding obstacles. When the drone arrives at the end of the room, it is considered to be a successful flight. When the drone collides with an object it is considered a crash, if it takes too long to complete the task (about 5 minutes of flying) it is considered to be stuck in a loop. The following algorithms are used to test its performance.

1) Kinect depth + Control: A simulated Kinect depth sensor is placed on the drone and this is fed into the Behaviour Arbitration control algorithm
2) Single Image depth + Control: The RGB image from the forward-facing camera of the drone is fed into the Single Image Depth CNN to get a depth map, which is then fed into the Behaviour Arbitration control algorithm

The performance of the different algorithms are compared in Table II. It should be noted that the simple template matching technique we used to detect corners was not feasible with the noisy depth estimation of the CNN, therefore we disabled that when evaluating the single image depth with control algorithm. As was to be expected, using single image depth results in lower performance than using perfect depth images from the kinect. It is noteworthy however that a significant portion of the failure cases were when the drone got stuck in a loop, meaning it did not actually collide into an object.

To further evaluate the performance of the depth estimation with the Behaviour Arbitration control algorithm, we set up two more virtual worlds. The first is an enclosed room with pole-like wall segments placed in the centre, which will be referred to as the pole-world. The other world is a maze-like corridor world, referred to as corridor world. The drone trajectories in these environments can be seen in figures 5 and 6. The simulated drone is spawned in different locations in these worlds and allowed to fly till it crashes. The mean over 10 flights in each world is shown in Table III).

Both the pole and corridor worlds have some texture on the floor, but no texture on the walls. Texture-less walls are a common feature of most indoor environments, and this is

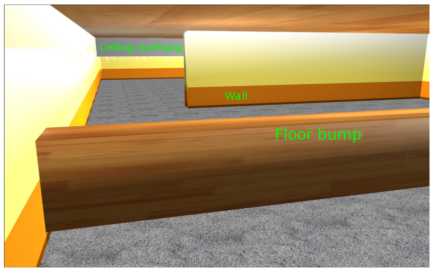

Fig. 4. Perspective view of one version (out of 100 simulated versions with different sizes of wall, ceiling overhang and floor bump) of the obstacle avoidance challenge.

TABLE III

PERFORMANCE OF DEPTH ESTIMATION WITH CONTROL ALGORITHM

World	Mean crash time [s]	Mean distance traveled [m]
Pole world	54.9	43.92
Corridor world	85.7	68.56

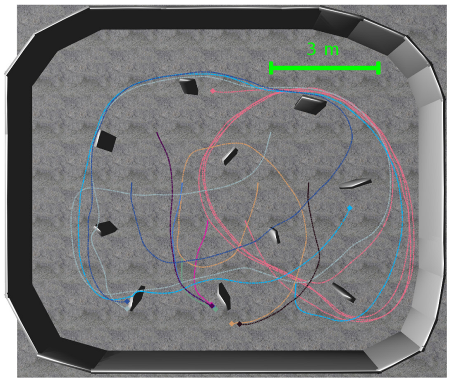

Fig. 5. Top-down view of the pole world with trajectories from simulated flights. Crashes are shown with a diamond.

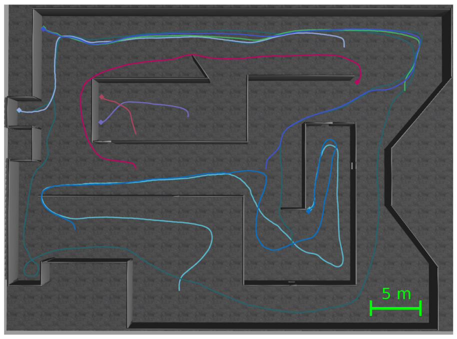

Fig. 6. Top-down view of the corridor world with simulated flight trajectories and crashes (diamonds at the end of trajectories).

where other, SfM based depth sensing methods are likely to fail.

We also do not fine-tune our network for these simulated environments. It is important to stress that the network trained on the NYU2 dataset (indoor images from American homes and not fine-tuned on the simulated data) is able to generate adequate depth maps for these scenarios. The Behaviour Arbitration control algorithm, using these depth maps, successfully steers the drone over tens of metres in both worlds.

C. Performance of the depth network in steering the drone in real indoor environments

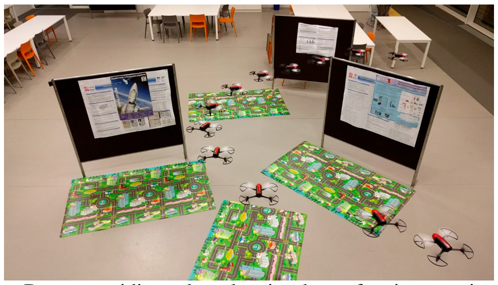

Fig. 7. Drone avoiding obstacles in the cafeteria experiment (9/10 successful flights).

We perform indoor experiments in our university environment. One set of experiments is performed at the department cafeteria, and another set in the department corridors. The cafeteria has large open spaces, where we put up poster screens as obstacles.

Figure 7 shows a trajectory of the drone while flying through the cafeteria. An experiment was determined to be a success if the drone successfully steered its way through 3 poster screens, which it did in 9/10 experiments.

In the corridor experiments, the drone is required to navigate safely up and down the narrower confines of the corridor (10 trials, 5 in each direction), turn (left/right) through a T-shaped junction (5 trials) and turn right at an L-shaped junction (5 trials). Figure 8 shows images captured from the drone camera and their corresponding depth maps. Steering commands ($\dot{\phi}$) are visualized using the red bar in the RGB image. The drone successfully navigated through the corridor and turned at the junctions in 18/20 flights.

The only parameters we adjusted for our experiments are two of the Behaviour Arbitration scheme, namely c_{obst} and λ_{avoid} from equation 3, which have an exponential and linear effect respectively. An informal interpretation of these two parameters is that when c_{obst} is smaller, the control algorithm will react to obstacles that are further away and increasing λ_{avoid} will make the drone turn away from an obstacle with a higher angular velocity. These parameters can be adjusted to either let the drone give a wider berth to obstacles, or get closer to them before turning away.

No explicit goal instructions are given to the drone ($f_{goto}(\phi)$ from equation 2 is turned off) in both cafeteria and corridor experiments.

The Parrot Bebop drone has problems holding position and drifts even when no control commands are given to it while operating over un-textured floors, as is often the case in indoor environments. Modifying the in-built position-hold algorithm (that utilizes the bottom-facing camera) was out of the scope of the experiments we conducted for this paper. We solved this problem by putting carpets under the flight path of the drone, which meant that the trajectory length was limited to the cumulative area of the carpets. The performance of the depth estimation and control algorithm over longer trajectories was evaluated qualitatively by walking down the corridors of the department with video footage captured on a mobile phone. These videos will be provided as supplementary material.

D. Failure cases

As the depth estimation is solely based on a single image and has no memory, it may fail in some situations where it lacks proper context. The known failure cases are discussed here.

a) Too close to wall: if the drone takes off very close to a wall so that it only sees this, it lacks the perspective view required to estimate the depth. In this case the control algorithm is unable to make a proper decision as to where to move to and would thus collide into the wall.

b) Corners: when flying around the drone may move towards a corner. While it is approaching the corner the depth map would indicate more free space in front than to the left or right, causing the drone to continue to move towards the corner.

c) Windows: Clear glass windows are a problem for the depth estimator as it sees right through them.

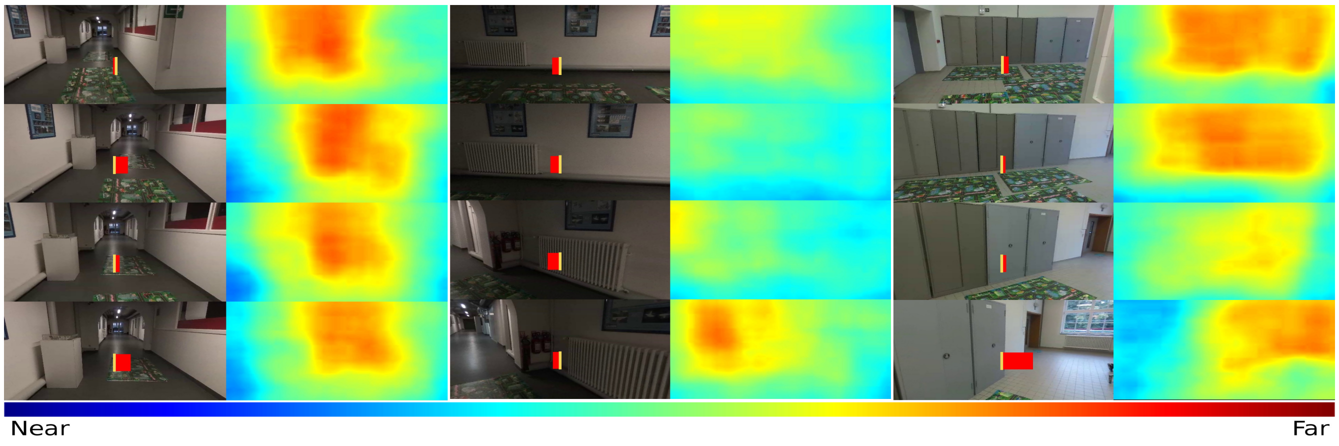

Fig. 8. Shown here are 3 experiments from the perspective of the drone, with RGB images and estimated depth maps. Red bar in the RGB image shows angular rate of change of yaw required to turn away from obstacles. The flights from left to right are as follows: flying down a corridor (9/10 successful flights), left turn at a T-junction (4/5 successful flights), Right turn at a bend in the corridor (5/5 successful flights)

All above problems can be solved using a cheap, low-range, low-power proximity sensor like sonar or a bump sensor on the propellor guards, which would also act as a final fail-safe on a production drone. We intend to incorporate sonar/bump sensing on our drone for future experiments, as it would allow us to fly more aggressively and also gather more training data for failure cases for the vision based algorithm, that could be fed back into the training regime.

Our trained CNN needs a laptop GPU to run in real-time, but can in future be incorporated into a small, low-power GPU like the NVIDIA Jetson [20] that is meant for drones and smartphones, yet built to support real-time deep learning applications.

VI. CONCLUSIONS

We demonstrate, in both simulated and real world experiments, the use of a deep CNN for single image depth estimation and a control algorithm based on this estimated depth, for steering a quadrotor away from obstacles in real time. We train our network with an online database of image and depth pairs and show that it is able to generalize over previously unseen indoor environments. We further fine-tune the network using data we collect ourselves, and observe qualitative and quantitative improvements in the generated depth maps.

In future work, we will experiment over a wider variety of indoor and outdoor environments and incorporate images from failure cases in the fine-tuning of the network.

REFERENCES

[1] P. Althaus and H.Christensen, "Behaviour coordination for navigation in office environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 2002, pp. 2298–2304.
[2] N. Gageik, P. Benz, and S. Montenegro, "Obstacle detection and collision avoidance for a uav with complementary low-cost sensors," *IEEE Access*, vol. 3, pp. 599–609, 2015.
[3] A. Barry and R. Tedrake, "Pushbroom stereo for high-speed navigation in cluttered environments," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3046–3052.
[4] H. Oleynikova, D. Honegger, and M. Pollefeys, "Reactive avoidance using embedded stereo vision for MAV flight," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 50–56.
[5] H. Alvarez, L. Paz, J. Sturm, and D. Cremers, "Collision avoidance for quadrotors with a monocular camera," in *Experimental Robotics*. Springer, 2016, pp. 195–209.
[6] S. Daftry, S. Zeng, A. Khan, D. Dey, N. Melik-Barkhudarov, J. A. Bagnell, and M. Hebert, "Robust monocular flight in cluttered outdoor environments," *arXiv preprint arXiv:1604.04779*, 2016.
[7] A. Saxena, H. Chung, and A. Ng, "Learning depth from single monocular images," in *Advances in Neural Information Processing Systems*, 2005, pp. 1161–1168.
[8] J. Michels, A. Saxena, and A. Ng, "High speed obstacle avoidance using monocular vision and reinforcement learning," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 593–600.
[9] I. Lenz, M. Gemici, and A. Saxena, "Low-power parallel algorithms for single image based obstacle avoidance in aerial robots," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 772–779.
[10] C. Bills, J. Chen, and A. Saxena, "Autonomous MAV flight in indoor environments using single image perspective cues," in *Robotics and automation (ICRA), 2011 IEEE international conference on*. IEEE, 2011, pp. 5776–5783.
[11] D.Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Advances in neural information processing systems*, 2014, pp. 2366–2374.
[12] F. Liu, C. Shen, G. Lin, and I. Reid, "Learning depth from single monocular images using deep convolutional neural fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015.
[13] M. Mancini, G. Costante, P. Valigi, and T. A. Ciarfuglia, "Fast robust monocular depth estimation for obstacle detection with fully convolutional networks," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 4296–4303.
[14] A. Giusti, J. Guzzi, D. Cireşan, F. He, J. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2016.
[15] J. B. S. Daftry and M. Hebert, "Learning transferable policies for monocular reactive mav control."
[16] N. Silberman, P. Kohli, D. Hoiem, and R. Fergus. (2012) Nyu depth dataset v2. [Online]. Available: http://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html
[17] A. Vedaldi and K. Lenc, "Matconvnet – convolutional neural networks for MATLAB," in *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.
[18] "Parrot bebop drone 2," http://global.parrot.com/au/products/bebop2/.
[19] "ROS driver for parrot bebop drone (quadrocopter) 1.0 & 2.0," http://bebop-autonomy.readthedocs.io/en/latest/.
[20] "The jetson tk1 embedded development kit," https://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html/.