Efficient UAV Autonomous Navigation with CNNs

Kamil Wojciech Mikolaj, Martin Lauersen, Tomer Tchelet, Daniel Ortiz Arroyo^{a,1}, and Petar Durdevic^{a,1}

***Abstract*—This paper presents a novel approach to the navigation of Unmanned Aerial Vehicles (UAV) for the autonomous inspection of wind turbines. Firstly, a Single Shot Detector (SSD) network is trained to detect wind turbines and their subcomponents. Then, an optimized template matching algorithm is used to estimate the distance between the UAV and the wind turbine, using as the template, the SSD bounding box prediction on the left image of a stereo camera. Lastly, an Extended Kalman Filter (EKF) estimates the position of the wind turbine's hub. The EKF is designed to compensate for CNN's latency while sending setpoints to the controller of the UAV.**

***Index Terms*—Visual Servoing Convolutional Neural Network Estimation Kalman filter**

I. Introduction

Wind turbine (WT) power generation with offshore wind energy is expected to show a 13% growth per year until 2040 [1]. As the total number of installed turbines continues to grow, the total maintenance costs will increase. It is estimated that WT inspection represents roughly 10% of their total maintenance costs [2]. Recently the use of manually piloted Unmanned Aerial Vehicles (UAVs) has become increasingly popular due to their low risk, but by fully automating the inspection task, the associated costs could be reduced. The autonomous inspection of WT can be split into three main tasks: navigation, inspection, and fault detection.

To navigate in 3D spaces approaches such as [3] use hybrid techniques using a map of the environment. Other approaches such as [4] focused on semi-autonomous drone inspection and damage detection in images, assuming that the drone was positioned in front of a WT's blade.

The authors in [5] used classical computer vision algorithms, such as edge detection and Hough transforms to localize a WT with a monocular camera. The distance estimation utilized a modified implementation of ORB-SLAM2 [6] that requires a very specific initialization procedure, limiting its use outside of a controlled environment. Similarly, [7] uses classical computer vision techniques to extract the main features of a WT. Distance estimation calculation assumed that the dimensions of the WT parts are known in advance. A Kalman filter was used to track the wind turbine hub location.

A more recent work [8] uses YOLOv2 [9], a convolutional neural network (CNN), to localize a WT on a pair of stereo images. Using the coordinates of the predicted pairs estimates, the distance from the UAV to a WT was calculated using stereo disparity.

[a]Authors are with the Department of Energy, Aalborg University, 6700 Esbjerg, Denmark

[1]{doa, pdl}@energy.aau.dk

CNNs have been shown to improve accuracy in object detection but they require vast amounts of labeled training data. In [10] it is shown that synthetically generated images can be used to train a neural network. Other studies such as [11], [12], use similar methods to generate datasets containing depth maps. These depth maps could then be used to train neural networks designed for depth estimation. However, rather than estimating the distance to a single object, these maps estimate the distance to all points in the scene, making them ill-suited for our application domain.

In this paper, we present an efficient system for the autonomous navigation of UAVs with applications in WT inspections. Our system employs CNNs together with a template matching scheme used for long-range distance estimation. The CNN was trained to detect WTs and their parts (blades, hub, tower, and nacelle) using an extended training dataset generated synthetically and labeled automatically.

We show that by using template matching and performing interpolation in the region of best match, the accuracy of the distance estimation at the maximum range can be increased with negligible computational overhead. We found experimentally the variances in the neural network bounding box prediction and our distance estimation method. An Extended Kalman Filter (EKF) fuses this information together with measurements from Inertial Measurement Unit (IMU) and Global Positioning System (GPS) to estimate the position of a detected WT target object in 3D space, while compensating for the time delay associated with CNN inference. These estimations are then used to control the navigation of an autonomous UAV toward a WT, with an unknown position. Our system was built and validated in a realistic simulation environment.

This paper is organized as follows, in Section II we briefly introduce the simulation environment we constructed for testing. Section III describes the CNN used, our image synthesis method, and training details. Section IV presents the template matching scheme, provides details on the EKF, and discusses how we characterized the CNN and distance estimator output for use in the EKF. Results are presented in Section V. Lastly, we present our conclusions in Section VI.

II. Simulation Platform

The Gazebo simulator was used to evaluate the methods proposed in this paper on the Robot Operating System (ROS).The simulation environment consists of a single WT and a UAV. To increase resemblance to the real world, ground texture, and moving skies were added to the simulation. The UAV was

TABLE I: Unmanned Aerial Vehicle (UAV) parameters used in the simulation

Parameter	Value	Unit
Weight	1.5	kg
Inertia xx, yy	0.013	kgm^2
Inertia zz	0.018	kgm^2
Max torque τ_x,τ_y	1	Nm
Max torque τ_z	1	Nm
Max lift	40	N

equipped with simulated stereo cameras, IMU and GPS. The final test environment can be seen in Figure 1.

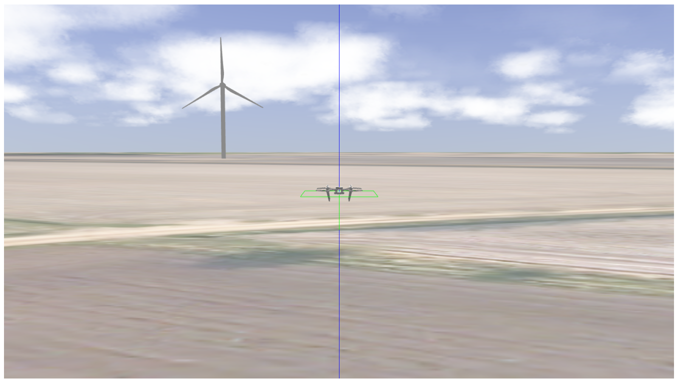
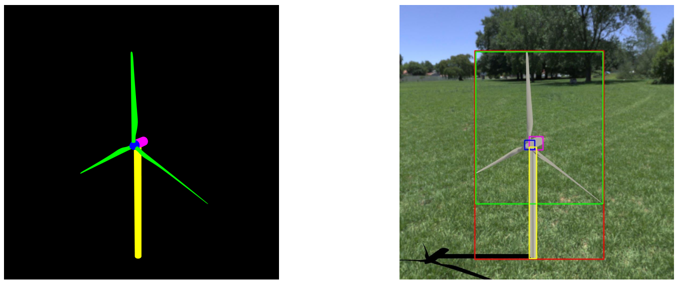


Fig. 1: A screenshot from the simulation environment where the quadcopter model is in the center with the WT model at a distance of 500 meters

To make the simulation as close to reality as possible, a digital 3D model of a real UAV quadcopter was designed as our development platform.The UAV model was simulated using the physical parameters given in Table I. The quadcopter has an IMU, GPS, and two identical cameras. Both GPS and IMU were simulated as ideal sensors with no noise, and 100 Hz sample rates to simplify the implementation of the low-level controllers and to easily obtain ground-truth measurements. However, random Gaussian noise was added to the IMU and GPS measurements before being passed to the `Kalman Filter` node. Additionally, rather than simulating the four individual rotors, the `Attitude Controller` node accepts ϕ_{sp}, θ_{sp}, ψ_{sp} angular velocities and z_{sp} altitude setpoints and applies the required τ_x, τ_y, τ_z $Thrust$ forces directly to the quadcopter body using parallel PID controllers. Similarly, the `Position Controller` node uses parallel PID controllers to generate ϕ_{sp}, θ_{sp}, and ψ_{sp} setpoints using the requested global x_{sp}, y_{sp} positions. Lastly, the `Setpoint Generator` node takes the position of the WT and calculates the offset location suitable to initiate the inspection step.

III. Deep Neural Network

The camera in the UAV was attached to the object detection (OD) CNN [13]. A variant of Single Shot Detector (SSD) network [14], the SSD300, was used as it offers a good compromise between inference time and accuracy. One of the problems of CNNs for OD is the need for a training labeled



(a) (b)

Fig. 2: Examples of synthetically generated dataset images with multiple IDs. IDs: Green = Blades, Yellow = Tower, Blue = Hub, Purple = Nacelle. (a) The image's corresponding segmentation map. (b) The generated training image with bounding boxes; a red box depicts the wind turbine class.

dataset with bounding boxes. However, manually labeling thousands of images is a very time-consuming process. To solve this problem an automatic labeling system was created to generate the dataset synthetically. The Unity game development platform and its machine learning image synthesis package was used for this purpose. This package utilizes the Depth Buffer in addition to several other Unity's built-in tools to generate segmentation maps of the image produced by the camera. The different segmentation maps extract features such as classes of objects, depth, and optical flow. In the context of this work, only the ID classification map, which exports a mask image with a unique color for each object ID in the frame, was used. These mask images, as seen in Figure 2, are not suitable for training an SSD model. To generate ground truth for bounding boxes, each image was traversed from top to bottom and right to left to find the first and last pixel belonging to each class. This operation yields a set of bounding boxes coordinates that were then saved in PASCAL VOC [15] format, which is suitable for training an OD neural network.

The scene in Unity was set up with the 3D model of a WT and a camera rotating around it with random distances, heights, and view angles. Additionally, to improve the generalization, 4 different WT models and 90 environments were alternated between frames. Further, an ID was specified for each subcomponent of the WT (Tower, Nacelle, Hub, and Blades), so the network could be able to detect smaller features when the WT extends beyond the picture frame.

The developed platform allows simple control of the resolution of the pictures and the size of the dataset. The final dataset was composed of $11,000$ images and their labels, $10,000$ for training, and $1,000$ for validation. The resolution of the images was chosen to be 300×300 as expected by the input layer of the SSD300 network. An example of a couple of synthetic images is shown in Figure 2.

In addition to the synthetic dataset, we used random data augmentation consisting of cropping, changing contrast, hue, saturation, swapping color channels, image expansion, and

TABLE II: Specifications of Raspberry Pi Camera Module V2 [17] as used for pinhole camera model

Specification	Nomenclature	Value
Video resolution	$w_{px} \times h_{px}$	1920 × 1080
Field of view (rad)	$\alpha_h \times \alpha_v$	$62.2\frac{pi}{180} \times 48.8\frac{pi}{180}$
Focal length (mm)	f	3.04
Pixel size (µm)	$P_x \times P_y$	1.12 × 1.12
Baseline (m)	b	0.3

horizontal flip. To train the CNN, we used the Adam optimizer [16] with dynamic learning rate scheduling as it provided faster convergence and a lower validation loss.

IV. Distance Estimation and Navigation

Once the CNN was trained and tested, the bounding box location information of the detected WT was utilized for navigation. To navigate to the WT, its position in relation to UAV's current position is calculated. This is done by using disparity on a stereo-image pair and applying template matching. The detected bounding box information together with depth estimation is used in conjunction with sensor data from IMU and GPS in an Extended Kalman filter (EKF) that estimates the position of the WT. This information is sent to the UAV's controller, which then navigates autonomously toward the estimated position.

A. Distance Estimation

Depth in 2D images can be calculated by finding the disparity between matching pixels across two corresponding images in stereo vision. For instance, the pixel disparity between the centers of the bounding boxes produced by two CNNs in both left and right images can be used to calculate the depth using the camera's parameters as was done in [8]. In our case, we used the pinhole camera model, with the specification set to be identical to the Raspberry Pi V2 camera [17]. The parameters shown in Table II were used. To convert disparity measurements into distance estimation, Equation 1 is applied.

$$d = \frac{b \cdot w_{px} \cdot \cot(\frac{\alpha_h}{2})}{2D_{px}} = \frac{477.42306}{D_{px}} \tag{1}$$

Where d is the distance estimate and b the baseline in meters, w_{px} the horizontal image resolution in pixels, α_h the horizontal field of view in radians, and D_{px} the disparity in pixels.

Disparity calculations need to produce multiple inferences which may introduce a significant delay in the system. To solve this issue, the bounding box data produced by a single CNN from a single camera's image is used as a template, then the object inside the template is localized on the other camera's image, using the template matching algorithm.

B. Template Matching

Template matching finds correlating objects in images. The method uses a template an object represented as a vector, that is slid across an image. Matching scores at each position are calculated as their similarity or distance in the vector space [18].

Our template matching method uses the bounding box prediction generated on the left image by the CNN. The template is scaled to full resolution and slid on the right image until it finds the best matching area. To optimize the matching operation, the template and the searched image are converted to grayscale. Further, as the cameras are placed in the same coordinate system with only a horizontal offset and are pointing in the same direction, template searches are simplified since rotated, scaled, and vertical searches are unnecessary. Furthermore, as a template from the left camera is matched to the image from the right camera, the search is limited to being performed only in one direction. Through experimentation, it was found that the CNN can only generate reliable detections down to a distance of $30m$. This is caused by several factors, one of which is that the simulated experimental platform (discussed in Section II) does not include textures on the models, rendering the WT undetectable. This limitation provides an upper bound for the search, restricting it to only 15 pixels, which can be obtained from Equation 1.

Lastly, the normalized correlation coefficient was used as the template matching method [19]. This method was chosen as it offers invariance to changes in global illumination and the normalized match scores ensure that the output is kept within an expected range (-1 to 1) between iterations.

Since the distance estimation is based on the disparity calculation it produces integer values. This generates a large error at large distances, which can be seen in Figure 3b. To solve this problem, we performed an interpolation to find sub-pixel disparity by fitting a second-order polynomial to the point with the best matching score and its two surrounding points, as shown in Figure 3a. This resulted in a much-improved distance estimation as can be seen in Figure 3b.

C. State Estimator

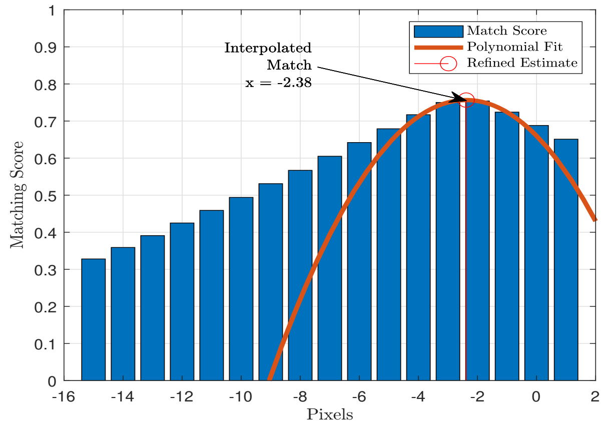
An EKF was used for this task with the state vector shown in Equation 2.

$$\boldsymbol{x_k} = [x_T \quad x \quad \dot{x} \quad \ddot{x} \quad y_T \quad y \quad \dot{y} \quad \ddot{y} \quad z_T \quad z \quad \dot{z} \quad \ddot{z}]^T \tag{2}$$

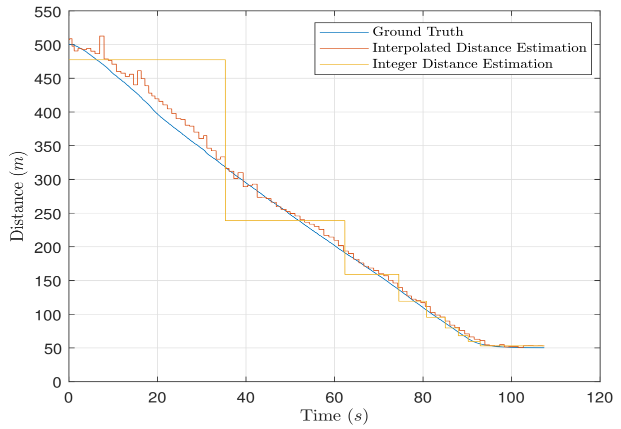
Where k denotes the discrete update steps and subscript T denotes the global position of the wind turbine. As angular data was not used in this project, the body and global frames are considered parallel at all times. Hence, the x-axis points forward, y-axis points left, and z-axis is pointing up in both frames.

The system matrix is identical on each axis, therefore a single axis system matrix $\boldsymbol{F_x}$ has the form:

$$\boldsymbol{F_x} = \begin{bmatrix} 1 & 0 & -\Delta t_k & -\frac{\Delta t_k^2}{2} \\ 0 & 1 & \Delta t_k & \frac{\Delta t_k^2}{2} \\ 0 & 0 & 1 & \Delta t_k \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$



(a)



(b)

Fig. 3: **(a)** An example of the interpolated sub-pixel disparity. **(b)** Comparison between the distances estimated by template matching (yellow), distances estimated with the interpolation (orange), and the ground truth (blue)

Such that:

$$\boldsymbol{F_k} = \begin{bmatrix} \boldsymbol{F_x} & & \\ & \boldsymbol{F_y} & \\ & & \boldsymbol{F_z} \end{bmatrix} \tag{4}$$

Where Δt_k is the update time of the filter.

Because the posterior distribution calculation is delayed as a result of the inference time of the neural network, an intermediate prediction step in the filter was added. To estimate the states at the time of capture, the system matrix F_i is used. Afterward, another system matrix F_r is used to make a prediction for the states at the time of the next update. This is shown in Equation 5.

$$\begin{aligned} \boldsymbol{x_c} &= \boldsymbol{F_i x_{k-1}} \\ \boldsymbol{\bar{x}_k} &= \boldsymbol{F_r x_c} = \boldsymbol{F_i F_r x_{k-1}} \\ \boldsymbol{\bar{P}_k} &= \boldsymbol{F_k P_{k-1} F_k^T} + \boldsymbol{Q_k} \end{aligned} \tag{5}$$

Where $\boldsymbol{x_c}$ denotes the estimated states at the current state of the system, $\boldsymbol{x_{k-1}}$ the posterior, $\boldsymbol{F_i}$ the system matrix with $\Delta t_i = Inference\,Time$ and $\boldsymbol{F_r}$ the system matrix with $\Delta t_r = \Delta t_k - \Delta t_i$ (i.e. equals the remaining time) and $\boldsymbol{Q_k}$ denotes the process covariance. As $\Delta t_i + \Delta t_r = \Delta t_k$, it can be shown mathematically that $\boldsymbol{F_i F_r} = \boldsymbol{F_k}$, proving the intermediate step does not affect the stability and the performance of the filter.

The system model is shown in Equation 6.

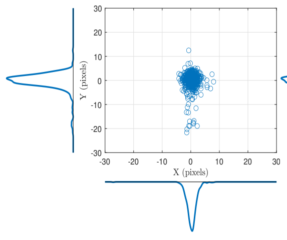
$$\begin{bmatrix} u_T \\ v_T \\ d \\ x \\ y \\ z \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = h(\boldsymbol{x}) = \begin{bmatrix} \frac{f(y_T - y)}{s_x\, P_x\, (x_T - x)} \\ \frac{f(z_T - z)}{s_y\, P_y\, (x_T - x)} \\ x_T - x \\ x \\ y \\ z \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} \tag{6}$$

Where u_T and v_T denote the center coordinates of the bounding box, d the distance estimation. Because the images are scaled before they are passed through the neural network, the pixel size used in the EKF has to be scaled as well. The variables $s_x = \frac{w_{px}}{300}$, $s_y = \frac{h_{px}}{300}$ denote that scaling factor. No modification was needed for the update step, therefore the standard EKF formulation is used as shown in Equation 7.

$$\begin{aligned} \boldsymbol{y_k} &= \boldsymbol{z_k} - h(\boldsymbol{\bar{x}_k}) \\ \boldsymbol{K_k} &= \boldsymbol{\bar{P}_k H_k^T}(\boldsymbol{H_k \bar{P}_k H_k^T} + \boldsymbol{R_k})^{-1} \\ \boldsymbol{H_k} &= \left.\frac{\delta h}{\delta \boldsymbol{x}}\right|_{\boldsymbol{\bar{x}_k}} \\ \boldsymbol{x_k} &= \boldsymbol{\bar{x}_k} + \boldsymbol{K_k y_k} \\ \boldsymbol{P_k} &= (\boldsymbol{I} - \boldsymbol{K_k H_k})\boldsymbol{\bar{P}_k} \end{aligned} \tag{7}$$

Where z_k and R_k denote the measurement vector and its covariance matrix, y_k the residual, K_k the Kalman gain, $h(\boldsymbol{\bar{x}_k})$ and H_k the measurement function and its Jacobian, and I is an identity matrix. While $\boldsymbol{z_k}$, the measurement vector, takes the distance estimation and the center coordinates of the bounding box, their respective variances need to be found. The following methods were used to find the characteristic variances for $[u_T\, v_T\, d]^T$.

- **Variances in Bounding Box Center** We trained the CNN to find the distribution of the error between u_t and v_t, the centers of the boxes detected by the neural network, and the centers of the ground truth boxes. Two experiments were conducted, one using the normalized confidences of each predicted box as the Probability Mass Function (PMF) of the center coordinates, to obtain an expected value for the center and one using the center of the box provided by the non-maximal suppression, a postprocessing step in SSD. As can be seen in Figure 4, both distributions are very similar, but distribution in 4a has a lower variance, which could be explained by the fact that the confidence of the detections concerns the classification and has no direct relation to the position of the box.


(a)

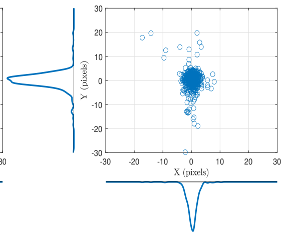

(b)

Fig. 4: Distribution of errors of the center of detected bounding boxes over 10,000 pictures. **(a)** Centers of the box from non-maximal suppression: $\sigma_x^2 = 3.997$ and $\sigma_y^2 = 15.108$. **(b)** Expected value for center coordinates using the normalized confidence of bounding boxes as Probability Mass Function (PMF): $\sigma_x^2 = 4.144$ and $\sigma_y^2 = 23.755$

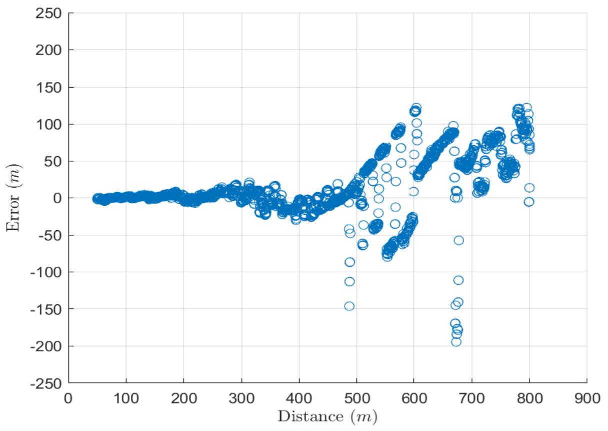


Fig. 5: Error in the estimation of distance as a function of the distance. Errors are less predictable at distances above 477 meters, which correspond to distances with disparity below one pixel.

- **Variance in Distance Estimation** The variance of the distance estimation was found by performing multiple experiments. The effect of displacement along the y- and z-axis on the distance estimation, was found to be negligible. This means that measurement d is independent of u_t and v_t. As opposed to that, the deviation of the estimation was found to depend on the distance. As can be seen in Figure 5, the error grows with the distance, meaning the variance in the estimation is not consistent. Therefore the standard deviation was approximated as a function of the distance estimation by fitting an exponential function to the variance of the distance estimations in increments of $50m$ as can be seen in Figure 6. The distance estimation statistics of u_t and v_t are listed in Table III.

V. Results

To test the complete system, the quadcopter was initially placed at the $(x = 0, y = 0, z = 10)$ point and the WT model at $(X = 500, y = 100, z = 0)$ meters. The initial

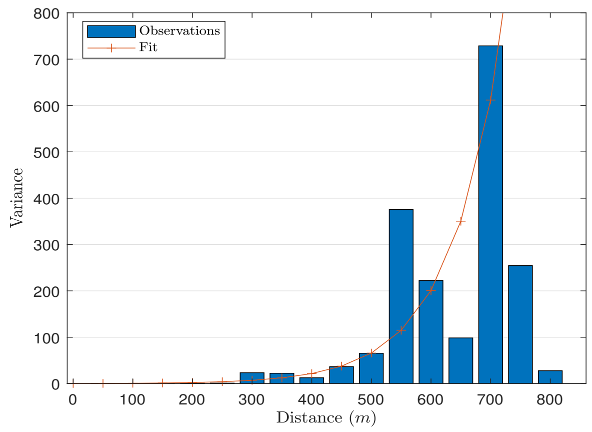


Fig. 6: Variance of distance estimation in $50m$ increments and fitting exponential function $0.2501e^{0.011146\,d}$.

TABLE III: Variances of neural network signals. u_T and v_T denote the x and y pixel coordinates identical to estimation

Signal (z)	Variance (σ^2)
u_T	$3.036\,px$
v_T	$8.116\,px$
d	$0.2501e^{0.011146\,d}\,m$

location was chosen to guarantee that the Wind Turbine (WT) falls within the camera's field of view, and that the distance is sufficiently large, ensuring that the initial disparity is less than one pixel.Once the state estimator was turned on, it started sending setpoints to the controller and the quadcopter autonomously navigated to a point 50 meters away from the WT at the height of the hub. This test was repeated 30 times, in order to ensure consistency in the results. The average RMS errors across these 30 runs can be seen in Table IV. Furthermore, the test was repeated with larger distances and showed successful results up to 650 meters, as long as the WT was visible in the frame from the initial point.

Additionally, tests were performed with the WT rotated and it was found that at certain angles the CNN is unable to detect it. At these angles the WT appears almost as a simple tower. The results of this test are shown in Fig 7. However, we noted that the performance varied between different WT models, both in distance and angle.

VI. Conclusion

The results described in Section V show that our method allowed the autonomous UAV to localize and navigate efficiently toward a WT in 3D space from any position, providing that the turbine is within the field of view of the camera and at a proper range. The use and modification of the EKF improved the predictions of the WT location and provided setpoints for a controller to autonomously navigate toward the target.

Several aspects of the system should be further investigated. In the current implementation, the final position of the UAV relative to the turbine depends on UAV's starting position. To

TABLE IV: Average Root Mean Square error across all 30 runs, with and without compensation. It can be seen that across the whole range of 500 meters, the compensation improves the accuracy of the position estimation on average by 6% on the x-axis, 10% on the y-axis and 30% on the z-axis.

		Full run	Within 450m	Within 100m
Compensated	x	10.8971	10.5201	5.7264
	y	9.3046	7.9134	0.5684
	z	14.6080	5.7807	3.3483
Uncompensated	x	11.635	11.8105	6.4039
	y	10.3554	9.5624	0.5869
	z	20.722	5.9762	3.2582

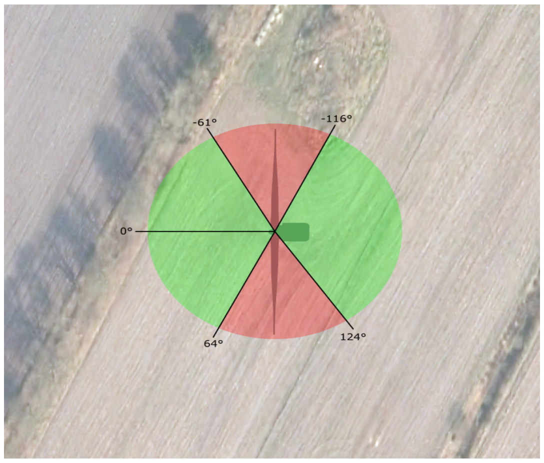


Fig. 7: Angle ranges of successful detection around the WT marked in green, no detections are marked in red.

navigate to a more specific point (e.g. directly in front of the turbine hub), information about the orientation of the wind turbine will be required.

Furthermore, our neural network model, while demonstrating potential in detecting WT, encounters difficulties when dealing with side views. This is not a concern if such a view occurs briefly during flight thanks to EKF tracking. However, if a WT is oriented this way with respect to the UAV starting location, the autonomous navigation cannot start. This shortcoming likely originates from the absence of detailed textures in our synthetic training dataset. To address this, future work should focus on improving the synthetic dataset by incorporating more realistic textures, aiming to enhance the model's performance from various perspectives."

Following this, another promising area for future work involves exploring generative models. These models could help create a synthetic, labeled training dataset by learning to replicate real-world textures and features, which can then be used to generate new, more authentic looking images.

References

[1] IEA, World Energy Outlook 2019, OECD Publishing, Paris, 2019. doi:10.1787/caf32f3b-en.

[2] R. Langan, B. H. Buck, Aquaculture Perspective of Multi-Use Sites in the Open Ocean: The Untapped Potential for Marine Resources in the Anthropocene, Springer Open, 2017. doi:10.1007/978-3-319-51159-7.

[3] D. Ortiz-Arroyo, A hybrid 3d path planning method for uavs, in: 2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS), 2015, pp. 123–132. doi:10.1109/RED-UAS.2015.7440999.

[4] M. Car, L. Markovic, A. Ivanovic, M. Orsag, S. Bogdan, Autonomous wind-turbine blade inspection using lidar-equipped unmanned aerial vehicle, IEEE Access PP (2020) 1–1. doi:10.1109/ACCESS.2020.3009738.

[5] R. Parlange, J. Martinez-Carranza, L. Sucar, B. Ren, Vision-based autonomous navigation for wind turbine inspection using an unmanned aerial vehicle, 2019.

[6] R. Mur-Artal, J. D. Tardós, ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras, CoRR abs/1610.06475 (2016). arXiv:1610.06475.
URL http://arxiv.org/abs/1610.06475

[7] M. Stokkeland, K. Klausen, T. A. Johansen, Autonomous visual navigation of unmanned aerial vehicle for wind turbine inspection, in: 2015 International Conference on Unmanned Aircraft Systems (ICUAS), 2015, pp. 998–1007.

[8] P. Durdevic, D. Ortiz-Arroyo, A deep neural network sensor for visual servoing in 3d spaces, Sensors 20 (5) (2020) 1437. doi:10.3390/s20051437.
URL http://dx.doi.org/10.3390/s20051437

[9] J. Redmon, A. Farhadi, Yolo9000: Better, faster, stronger, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[10] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, S. Birchfield, Training deep networks with synthetic data: Bridging the reality gap by domain randomization, CoRR abs/1804.06516 (2018). arXiv:1804.06516.
URL http://arxiv.org/abs/1804.06516

[11] J. Zbontar, Y. LeCun, Stereo matching by training a convolutional neural network to compare image patches, J. Mach. Learn. Res. 17 (2016) 65:1–65:32.

[12] G. Huang, Y. Gong, Q. Xu, K. Wattanachote, K. Zeng, X. Luo, A convolutional attention residual network for stereo matching, IEEE Access 8 (2020) 50828–50842.

[13] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, J. Walsh, Deep learning vs. traditional computer vision, in: K. Arai, S. Kapoor (Eds.), Advances in Computer Vision, Springer International Publishing, Cham, 2020, pp. 128–144.

[14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, A. C. Berg, SSD: single shot multibox detector, CoRR abs/1512.02325 (2015). arXiv:1512.02325.
URL http://arxiv.org/abs/1512.02325

[15] P. VOC, http://host.robots.ox.ac.uk/pascal/VOC/voc2008/htmldoc/, accessed: 2020-09-05.

[16] D. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv.org (2017).

[17] R. P. Foundation, Raspberry pi v2 camera module documentation, https://www.raspberrypi.org/documentation/hardware/camera/, accessed: 2020-04-14.

[18] R. Brunelli, Template Matching Techniques in Computer Vision: Theory and Practice, John Wiley and Sons, 2009. doi:10.1002/9780470744055.

[19] N. S. Hashemi, R. B. Aghdam, A. S. B. Ghiasi, P. Fatemi, Template matching advances and applications in image analysis (2016). arXiv:1610.07231.