Robotics: Science and Systems 2021
Held Virtually, July 12–16, 2021

EVPropNet: Detecting Drones By Finding Propellers For Mid-Air Landing And Following

Nitin J. Sanket[1], Chahat Deep Singh[1], Chethan M. Parameshwara[1], Cornelia Fermüller[1], Guido C.H.E. de Croon[2], Yiannis Aloimonos[1]

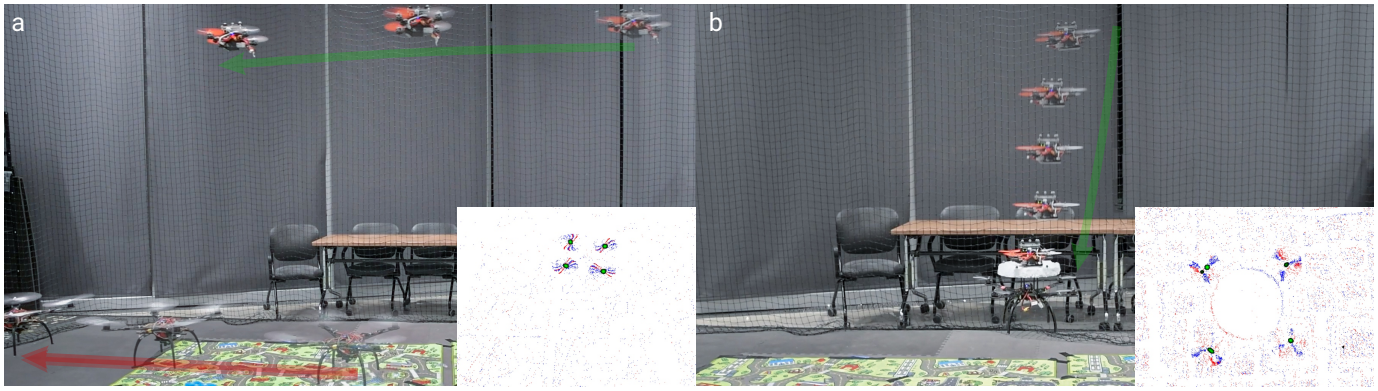

Figure 1. Applications presented in this work using the proposed propeller detection method for finding multi-rotors. (a) Tracking and following an unmarked quadrotor, (b) Landing/Docking on a flying quadrotor. Red and green arrows indicates the movement of the larger and smaller quadrotors respectively. Time progression is shown as quadrotor opacity. The insets show the event frames $\mathcal{E}$ from the smaller quadrotor used for detecting the propellers of the bigger quadrotor using the proposed *EVPropNet*. Red and blue color in the event frames indicate positive and negative events respectively. Green color indicates the network prediction. All the event images in this paper follow the same color scheme. Vicon estimates are shown in corresponding sub-figures of Fig. 8. *All the images in this paper are best viewed in color on a computer screen at a zoom of 200%.*

***Abstract*—The rapid rise of accessibility of unmanned aerial vehicles or drones pose a threat to general security and confidentiality. Most of the commercially available or custom-built drones are multi-rotors and are comprised of multiple propellers. Since these propellers rotate at a high-speed, they are generally the fastest moving parts of an image and cannot be directly "seen" by a classical camera without severe motion blur. We utilize a class of sensors that are particularly suitable for such scenarios called event cameras, which have a high temporal resolution, low-latency, and high dynamic range.**

In this paper, we model the geometry of a propeller and use it to generate simulated events which are used to train a deep neural network called *EVPropNet* to detect propellers from the data of an event camera. *EVPropNet* directly transfers to the real world without any fine-tuning or retraining. We present two applications of our network: (a) tracking and following an unmarked drone and (b) landing on a near-hover drone. We successfully evaluate and demonstrate the proposed approach in many real-world experiments with different propeller shapes and sizes. Our network can detect propellers at a rate of 85.1% even when 60% of the propeller is occluded and can run at upto 35Hz on a 2W power budget. To our knowledge, this is the first deep learning-based solution for detecting propellers (to detect drones). Finally, our applications also show an impressive success rate of 92% and 90% for the tracking and landing tasks respectively.

Supplementary Material

The accompanying video and code are available at http://prg.cs.umd.edu/EVPropNet.

[1]Perception and Robotics Group, University of Maryland Institute for Advanced Computer Studies, University of Maryland, College Park.
[2]Micro Air Vehicle Laboratory, Delft University of Technology.
Corresponding author: Nitin J. Sanket.

I. Introduction

Aerial robots or drones have become ubiquitous in the last decade due to their utility in various fields such as exploration [1, 2, 3, 4], inspection [5], mapping [6], search and rescue [7, 8], and transport [9]. The low-cost and wide availability of commercial drones for photography, agriculture and hobbying has skyrocketed drone sales [10]. This has also given rise to a series of malicious drones which threaten the general security and confidentiality. This necessitates the detection of drones. To make this problem hard, drones come in various shapes and sizes and generally do not carry any distinct visual on them to make them easy for visual detection. To this end, we propose to detect an unmarked drone by detecting the most ubiquitous part of a drone – the propeller. It is serendipitous that most of the common drones are multi-rotors and have more than one propeller, making their detection using the proposed approach easier. Detecting propellers is a daunting task for classical imaging cameras since it would require an extremely short shutter time and high sensitivity which make such sensors expensive and bulky. A class of sensors designed by drawing inspiration from nature that excel at the task of low-latency and high-temporal resolution data are called event cameras [11, 12]. Recent advances in sensor technologies have increased the spatial resolution of these sensors by about 10× in the last 5 years [13]. These event cameras output per-pixel temporal intensity differences caused by relative motion with microsecond latency instead of traditional images frames. We utilize the fact that propellers are moving much faster than any other part of the scene. The problem formulation and our

contributions are described next.

A. Problem Formulation and Contributions

An event camera (moving or stationary) is looking at a flying drone with at-least one spinning propeller and our goal is to locate the propeller's center on the image plane. A summary of our contributions are:

- We simplify the geometric model of a propeller for the projection on the image plane which is used to generate event data.
- A deep neural network called *EVPropNet* trained on the simulated data which generalizes to the real-world without any fine-tuning or re-training for different propellers.
- Two specific applications of our *EVPropNet*: (a) Tracking and following an unmarked moving quadrotor (Fig. 1a), (b) Landing on a near-hover quadrotor (Fig. 1b) evaluated with on-board computation and sensing.
- Finally, we make our network `EdgeTPU` optimized so that it can run at 35Hz with a power budget of just 2W enabling deployment on a small drone.

B. Related Work

We subdivide the related work into three parts: detection of an unmarked drone based on appearance (on a classical camera), detection of a marked collaborative drone and detection of moving segments using event cameras.

1) Appearance based drone detection:

Classical RGB image based drone detection is an instance of object detection and has been accomplished by methods like Haar cascade detectors, with the newer deep learning based detectors such as YoLo [14] topping the accuracy charts [15, 16, 17]. One can clearly see that these methods work well when the drone is large in the frame and against a bright sky, thereby detecting the contour of the drone from its silhouette. Pawełczyk *et al.* [15] show extensive results on how the state-of-the-art appearance based drone detectors fail when the drones are against a non-sky background (such as trees which are very common).

2) Marked collaborative drone detection:

Marked drones are detected using a set of fiducial markers either for leader-follower configurations [18], swarming behaviors [19] or for docking [20]. Most commonly, a visual fiducial marker based on April Tags [21] or CC-Tags [22] is used for these tasks due to their robustness and near-invariance to angles. Moreover, they also provide the ability to distinguish between different tags which are particularly useful for tracking multiple drones. Li *et al.* [20] designed a custom tag similar to the CC-Tag and showed that it can be used for precise control for docking. On the contrary, Walter *et al.* [18] demonstrated the usage of Ultra-Violet (UV) spectrum which is robust to changing environmental conditions such as changing illumination and the presence of undesirable light sources and their reflection.

3) Moving Object Segmentation Using Event Cameras:

Event cameras, as described earlier are tailor made for detecting the parts of the image which have motion different to that of the camera (this task is commonly called motion segmentation). Mitrokhin *et al.* [23] developed one of the first motion segmentation frameworks using event cameras for challenging lighting scenes highlighting the efficacy of event cameras to work at high-dynamic range scenes for fast moving objects. Stoffregen *et al.* [24] introduced an Expectation-Maximization scheme for segmenting the motion of the scene into various parts which was further improved in-terms of speed and accuracy by Parameshwara *et al.* [25] by proposing a motion propagation method based on cluster keyslices. The concept of motion segmentation has also been deployed on quadrotors for detection of other moving objects (including other unmarked drones) with a monocular [26] and a stereo event camera [27].

C. Organization of the paper

We first describe a geometric model of the propeller and then derive a simplified model of it's image projection in Sec. II. The geometric model is then used generate event data to train the proposed *EVPropNet* as described in Sec. III. We then present two applications of our network: (a) Tracking and following an unmarked drone and (b) Landing on a near-hover drone in Sec. IV. We then present extensive quantitative evaluation of our network and applications along with qualitative results on different real world propellers in Sec V. Finally, we conclude the paper in Sec. VI with parting thoughts on future work.

II. Geometric Modelling of a Propeller

We first discuss a geometric model of a propeller [28] and then describe how it's projection can be approximated with a set of cubic basis splines. A propeller's spine is constructed by rotating a straight line on a helicoidal surface. The coordinates of a point $\mathbf{x}$ on a surface formed by a straight line rotating about the X axis and concurrently moving along this axis is given by

$$\mathbf{x} = \begin{bmatrix} \frac{p\phi}{2\pi} & r\sin\phi & r\cos\phi \end{bmatrix}^T \tag{1}$$

Where p is the pitch of the propeller, r is the radius and ϕ is the angle of rotation in YZ plane of the radius arm relative to the Z^W axis (Fig. 2a, also see Table I for a tabulation of the parameters used in this derivation). Note that, p here refers to the nose-tail pitch as it is the most common definition used by manufacturers. Now, the locus of the mid-chord points of a rotating right handed propeller blade with $\phi = 0$ initially is given by

$$\mathbf{x}_{c/2} = \begin{bmatrix} -\left(i_G + \frac{p\theta_S}{2\pi}\right) & -r\sin\left(\phi - \theta_S\right) & r\cos\left(\phi - \theta_S\right) \end{bmatrix}^T \tag{2}$$

Here, θ_S (Figs. 2b and 2c) denotes the skew angle and is defined as the angle between the line normal to the shaft

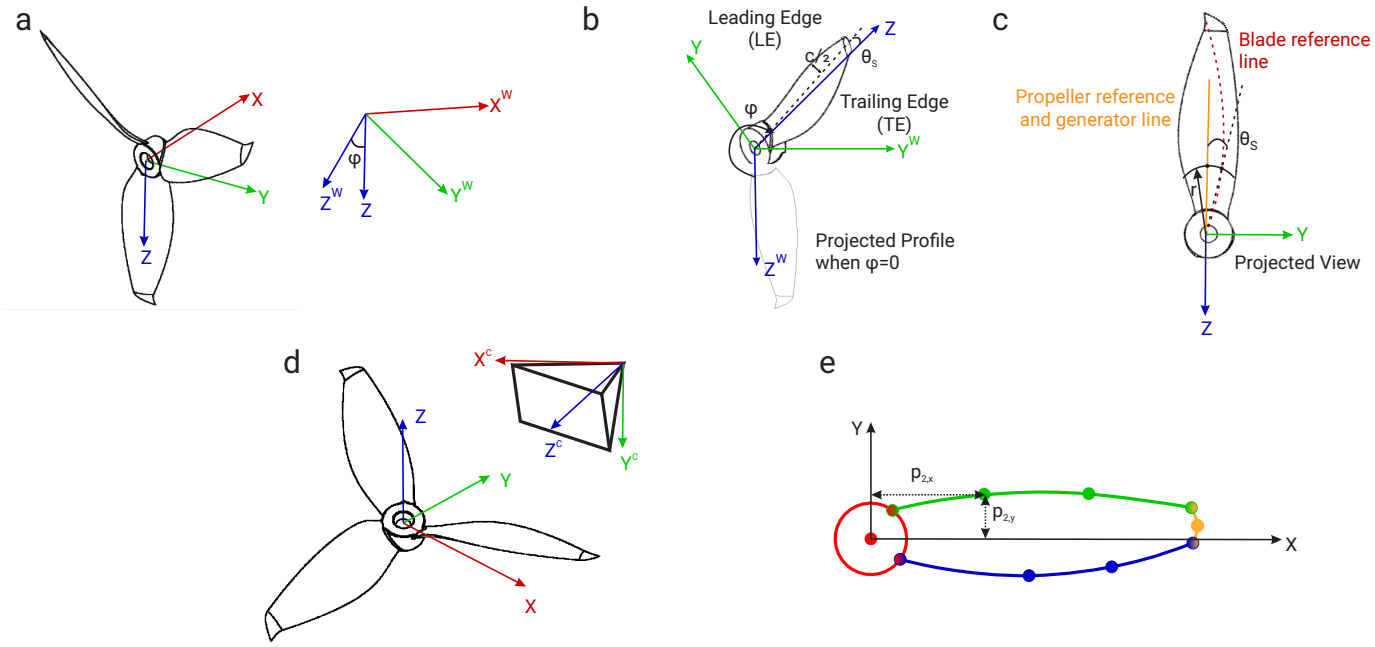

Figure 2. (a) Coordinate frames used for the geometric modelling of a propeller, (b) Blade coordinate definition, (c) Skew definition, (d) Coordinate axes for propeller projection on camera, and (e) Simplified model of the projection of the propeller blade; Each color represents a single spline and points with same color denote knots used to fit the cubic spline. Bi-color points are used as knots for both the splines of respective color. See Table I for a tabulation of the variables used in this figure.

axis (called *directrix* or *propeller reference line*) and the line drawn through the shaft center line and the mid-chord point on the projected image of the propeller looking normally along the shaft center line and i_G denotes the generator line rake (distance that is parallel to the X-axis, from the directrix to the point where the helix of the section at radius r cuts the $X-Z$ plane). Extending Eq. 2 for the leading and trailing edges of the blade (Fig. 2b) gives us

$$\mathbf{x}_{LE/TE} = \begin{bmatrix} -\left(i_G + \dfrac{p\theta_S}{2\pi} + \dfrac{c}{2}\sin\theta\right) \\ -r\sin\left(\phi - \theta_S \pm \dfrac{90c\cos\theta}{\pi r}\right) \\ r\cos\left(\phi - \theta_S \pm \dfrac{90c\cos\theta}{\pi r}\right) \end{bmatrix} \tag{3}$$

Here $\theta = \tan^{-1}\left(\frac{p}{2\pi r}\right)$ is the pitch angle, c is the chord length at a certain radius and π and θ_S are in $^\circ$.

The above set of equations define how the propeller's *nose-tail line* can be generated in 3D. However, the blade section geometry is an aerofoil with a top and a bottom surface. A point on the top and bottom surfaces are given by

$$\mathbf{x}_{T/B} = \begin{bmatrix} x_c \mp y_t \sin\psi & y_c \pm y_t\cos\psi \end{bmatrix}^T \tag{4}$$

where y_c is the y offset from the chord line, y_t is the ordinate of the point in question and ψ is the slope of the chamber line at the non-dimensional chordal position x_c. Now, if we consider the chord's mid point as the local origin, then a point's coordinates $\mathbf{x}$ are given by

Table I
PARAMETERS USED IN GEOMETRIC MODEL OF THE PROPELLER.

Parameter Notation	Parameter Description
p	Propeller Pitch (nose-tail)
r	Radius
ϕ	Angle of rotation of radius arm relative to Z^W in YZ plane
i_G	Generator line rake
θ_S	Skew angle
c	Chord length
ψ	Chamber line slope at x_c
Subscripts T and B	Top and Bottom surfaces of blade
Subscripts LE and TE	Top and Bottom edges of blade
K	Camera intrinsic matrix
f	Camera focal length
c_x and c_y	Camera principal points
$\mathbf{p_i}$	Spline control points
$N_{i,k}(t)$	Spline basis function

$$\mathbf{x} = \begin{bmatrix} -\left(i_G + \dfrac{p\theta_S}{2\pi}\right)(0.5c - x_c)\sin\theta + y_{u,B}\cos\theta \\ r\sin\left(\theta_S - \dfrac{180\left((0.5c - x_c)\cos\theta - y_{u,B}\sin\theta\right)}{\pi r}\right) \\ r\cos\left(\theta_S - \dfrac{180\left((0.5c - x_c)\cos\theta - y_{u,B}\sin\theta\right)}{\pi r}\right) \end{bmatrix} \tag{5}$$

Where $y_u = y_c \pm y_t\cos\psi$ (Eq. 4). To convert x into global coordinates $\mathbf{x}^W$,

$$\mathbf{x}^W = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \mathbf{x} \tag{6}$$

where ϕ is the angle between two adjacent blades. Although the propeller thickness varies along its chord line, we can neglect this value since we are concerned with the projection of the propeller on the image plane assuming that distance from

the camera is $\gg$ propeller thickness. We want to simulate how a propeller would "look" when imaged from a camera (which would be later converted into an event stream). We assume that the image is captured from a calibrated camera formulated using the pinhole model given by

$$\mathbf{x} = K\begin{bmatrix} R, & T \end{bmatrix}\mathbf{X}; \quad K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

where K is the camera calibration matrix, f is the focal length and c_x, c_y denotes the principle point, $\begin{bmatrix} R, & T \end{bmatrix}$ denotes the pose of the camera, $\mathbf{X}$ is the world point being imaged and $\mathbf{x}$ is the location of the point on the image plane (with reuse of variables). $\mathbf{X}$ in Eq. 7 (see Fig. 2d for definition of coordinate frames) is given by $\mathbf{x}^W$ from Eq. 6. Although, this method is the most generative way to model a propeller, i.e., generate 3D points of the propeller and then project onto the image plane, it would be computationally very expensive for a high fidelity image, hence we approximate the projection of a propeller blade with a set of cubic basis splines [29, 30] described next. Let the $n+1$ control points be $\mathbf{p_0}, \cdots, \mathbf{p}_n$ and $m+1$ knot vectors be $\{t_0, \cdots, t_m\}$, the spline curve $s(t)$ of degree k is given by

$$s(t) = \sum_{i=0}^{n} \mathbf{p}_i N_{i,k}(t) \tag{8}$$

Here, $N_{i,k}(t)$ is the basis function of degree k and is computed recursively as

$$N_{i,0}(t) = \begin{cases} 1 & \text{if } t_i \leq t \leq t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} N_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} N_{i+1,k-1}(t) \tag{10}$$

In particular, $m = n + k + 1$ and we utilize the uniform B-spline, i.e., all the knots are uniformly distributed and are evaluated using the procedure described in [31]. We model each propeller blade using 4 cubic B-splines: one for the hub, one for the top part of the blade, one for the bottom part of the blade and one for the tip of the blade (Fig. 2e). Each blade is replicated at a uniform angular spacing for the required number of blades (i.e., for a 3 bladed propeller, the blades would have an angle of 120° between them).

III. EVPropNet

We will now discuss how the geometric model of the propeller is used to generate event data. Then we describe the network architecture and loss function used to train *EVPropNet*.

A. Event Generation

As explained earlier, we now have a single image of a propeller with the required number of blades at the required high resolution. We overlay this propeller image on top of a random real image background from the MS-COCO dataset

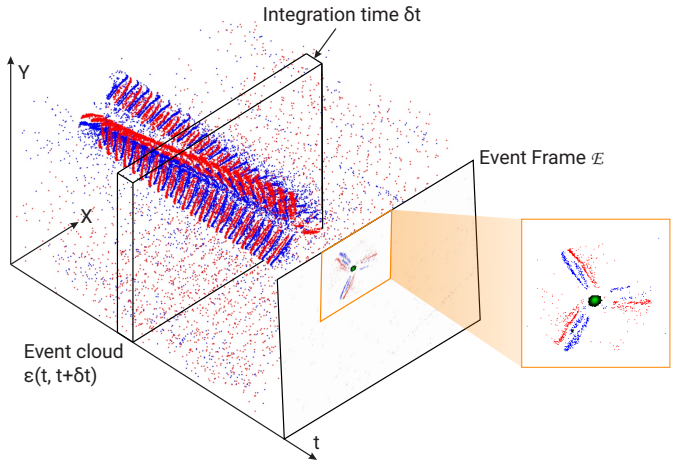

Figure 3. Spatio-temporal event cloud $\mathcal{E}$ and Event frame $\mathcal{E}$. The cloud shows that the propeller creates a helix in the spatio-temporal domain. The zoomed in view shows the propeller with positive events colored red and negative events colored blue along with network prediction as green with the color saturation indicating confidence.

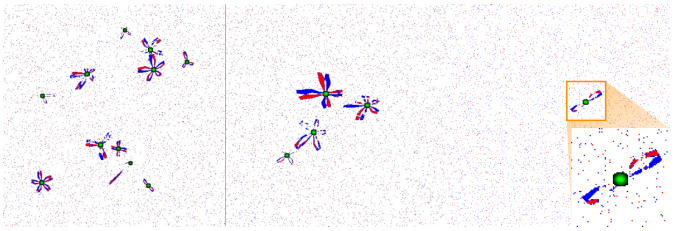

Figure 4. Sample event images $\mathcal{E}$ from the generated synthetic dataset used to train *EVPropNet*. Here red and blue colors show positive and negative events respectively. Green color indicates our ground truth label with the color saturation indicating confidence as defined by Eq. 14.

[32] at a random starting angle θ_{HB} (angle the propeller reference line makes with the propeller Y axis), we denote this as image $\mathcal{I}_t$. We then perform the same procedure for a $\theta_{HB} + \delta\theta$ angle (with the same background) to generate the image $\mathcal{I}_{t+\delta t}$. Here, $\delta\theta = \omega\delta t$ is the angle the blade would rotate depending on the rotational speed of the propeller ω and the event frame integration time δt. We use a simple model for the event camera and events are triggered at a location $\mathbf{x}$ when

$$\|\log(\mathcal{I}_t(\mathbf{x})) - \log(\mathcal{I}_{t+\delta t}(\mathbf{x}))\|_1 \geq \tau \tag{11}$$

The event stream/cloud $\mathcal{E}$ is represented by

$$\mathcal{E} = \{\begin{bmatrix} \mathbf{x} & t & \text{sgn}(\log(\mathcal{I}_t(\mathbf{x})) - \log(\mathcal{I}_{t+\delta t}(\mathbf{x}))) \end{bmatrix}^T\} \tag{12}$$

Where τ is a user defined threshold and $\mathbf{x}$ is the pixel location. $\mathcal{E}$ is called the event cloud which is used to create the so-called *Event-frame* $\mathcal{E}$ (Fig. 3 shows how $\mathcal{E}$ and $\mathcal{E}$ look) which is used as the input to the network.

$$\mathcal{E} = \text{sgn}(\mathbb{E}_t(\text{Pol}(\mathcal{E}(t, t+\delta t)))) \tag{13}$$

Here, $\mathbb{E}_t$ denotes the averaging operator only in time axis, Pol denotes the polarity values are extracted per pixel (last row of each element of $\mathcal{E}$).

B. Data Generation

We generate 10K event frames $\mathcal{E}$ for training our network (See Fig. 4 for sample images with labels overlaid). Each event frame contains upto N propellers (set to 12 in our

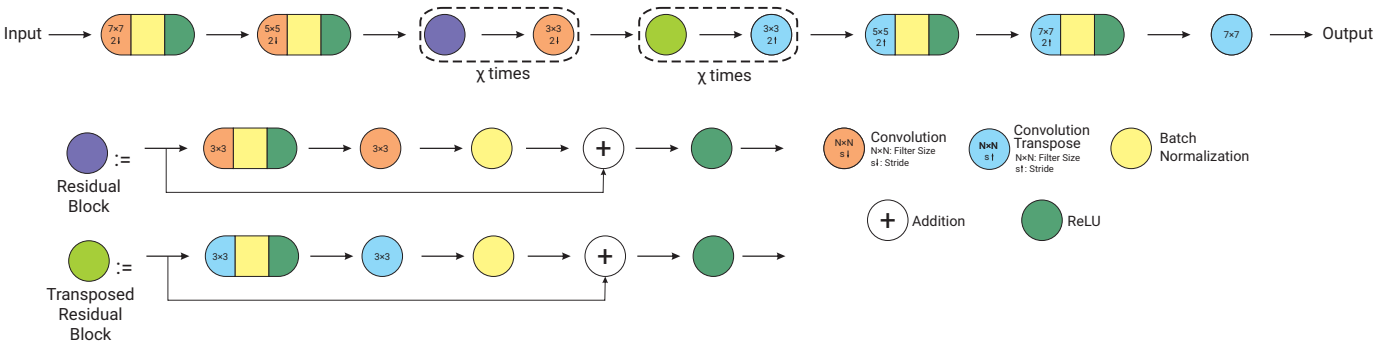

Figure 5. Network architecture for *EVPropNet* (χ is a hyperparameter along with expansion rate – rate at which the number of neurons grow after each block). If no down/up-sampling rate is shown, it is taken to be 1. *This image is best viewed on the computer screen at a zoom of 200%.*

case) with number of blades per propeller randomized from 2 to 6. The events for each propeller are obtained by varying τ as a gaussian random variable to provide some randomness in data generation along with randomization of the color of the propeller in $\mathcal{I}_t$ (same color is used for $\mathcal{I}_{t+\delta t}$) along with varying ω (rotational speed of the propeller, this is equivalent to varying the integration time δt). We also vary the background image for every propeller from the MS-COCO dataset. Each propeller is also warped using a random homography matrix to account for different camera angles along with scaling them (setting the pixel size of the propeller in the event image) to account for distance variation from the camera. Finally, we also vary the shape of each propeller by varying the basis spline parameters (to include bullnose and normal type propellers as well). See Fig. 4 for some sample images from the dataset used to train *EVPropNet*. Note that, we do not use an event simulator like ESIM[33] to generate events since we only require $\mathcal{I}_t$ and $\mathcal{I}_{t+\delta t}$ which are directly constructed, hence this process is multiple orders of magnitude faster than real-time and parallelized.

C. Network Architecture and Loss Function

We choose an encoder-decoder architecture based on the ResNet [34] backbone (Fig. 5) as it has the best accuracy and speed tradeoff [35, 36] with 2.7M parameters and 10MB model size. We train our network using simple mean square loss $\mathcal{L} = \mathbb{E}\left((\hat{p} - \tilde{p})^2\right)$ between the ground truth $\hat{p}$ and prediction $\tilde{p}$. $\hat{p}$ is obtained by Gaussian smoothing the perfect label $\hat{p_0}$ (binary mask) as given by Eq. 14 (σ is the variance) to account for small distortion introduced by approximation of propeller shape. This approach is similar to the one introduced in [37].

$$\hat{p} = \frac{1}{2\pi\sigma^2} e^{-\left(\|\hat{p_0}\|_2 / 2\sigma^2\right)} \tag{14}$$

We choose the number of residual and transposed residual blocks χ as 2 and expansion factor as 2 (factor with which number of neurons grow after every block in Fig. 5).

Finally, *EVPropNet* was trained with a learning rate of 1e-4 using ADAM optimizer with a batch size of 32 for 50 epochs.

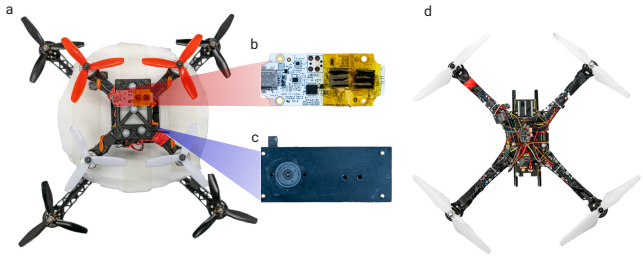

Figure 6. (a) Smaller quadrotor on the bigger quadrotor used for landing experiments (Sec. IV-A), (b) Gutted Coral USB Accelerator with custom heat sink used to run the neural networks, (c) Samsung Gen 3 DVS sensor used for experiments, (d) Bigger quadrotor used in the following experiments (Sec. IV-B).

IV. Applications

We describe two applications of our propeller detection, i.e., following an unmarked moving quadrotor and landing on a near-hover quadrotor.

A. Following

In this application, the goal is to track and follow a quadrotor (or a multirotor in general) either for swarming or reconnaissance purposes. We detect the quadrotor as the centroid of filtered propeller detections as described in Sec. IV-C. The control policy for altitude is to maintain the area of the quadrilateral joining the propeller points constant and the control policy for roll and pitch is to maintain the centroid of the quadrilateral in the center of the image and are given by:

$$\mathbf{u}_\phi(t) = K_{p,\phi} e_x(t) + K_{i,\phi} \int_0^\tau e_x(\tau) d\tau + K_{d,\phi} \frac{de_x(t)}{dt} \tag{15}$$

$$\mathbf{u}_\theta(t) = K_{p,\theta} e_y(t) + K_{i,\theta} \int_0^\tau e_y(\tau) d\tau + K_{d,\theta} \frac{de_y(t)}{dt} \tag{16}$$

$$\mathbf{u}_T(t) = K_{p,T} e_A(t) + K_{i,T} \int_0^\tau e_A(\tau) d\tau + K_{d,T} \frac{de_A(t)}{dt} \tag{17}$$

Where e_x and e_y denote the difference between detected quadrilateral center on the image plane and the center of the image and e_A denotes the difference between area to maintain and current area.

B. Landing

For the second application, the goal is to land on a near-hover quadrotor either for in-air battery switching or

infiltration of a hostile drone. We utilize the following key observations from [38]:

- The quadrotor flying above experiences negligible aerodynamic disturbances from mutual interaction.
- Forces in direction normal to the downwash are negligible and those in the downwash direction are significant.
- The aerodynamic torques disturb the bottom quadrotor so that it aligns vertically with the top quadrotor.

The smaller quadrotor (which will land) explores the area for any other quadrotor (or any multirotor in general), once it detects a multirotor, it switches to the align maneuver, where we perform the following control policy for roll ϕ and pitch θ axes (X and Y respectively) for aligning the center of the detected quadrotor (centroid of filtered propeller detections as described in Sec. IV-C) with the center of the image:

$$\mathbf{u}_{\phi}(t) = K_{p,\phi}e_x(t) + K_{i,\phi}\int_0^{\tau} e_x(\tau)d\tau + K_{d,\phi}\frac{de_x(t)}{dt} \quad (18)$$

$$\mathbf{u}_{\theta}(t) = K_{p,\theta}e_y(t) + K_{i,\theta}\int_0^{\tau} e_y(\tau)d\tau + K_{d,\theta}\frac{de_y(t)}{dt} \quad (19)$$

Where e_x and e_y denote the difference distance between detected quadrotor center on the image plane and the center of the image. Once the errors e_x and e_y are lower than a threshold, we decrease altitude at a constant rate, checking for x and y alignment at every control loop and re-aligning as necessary. Once we are close to the big quadrotor (on which the smaller quadrotor will land), we initiate the land command.

C. Quadrotor Location from Detected Propellers and Filtering

We filter each propeller location on the image plane using a linear Kalman filter [39]. The motion model is a constant optical flow model. Once we obtain detections with a confidence above a certain threshold, the filtered propeller locations are used to compute the centroid of the quadrilateral (for the quadrotor case, polygon in general) which is used for control (to compute e_x and e_y), along with the area for altitude control.

V. Experimental Results and Discussion

A. Quadrotor Setup

All our experiments are performed with quadrotors for their minimal hardware complexity and cost-effectiveness but they can directly be adapted to any multirotor vehicle. Our smaller quadrotor is a custom built platform on a QAV-X 210mm sized (motor center to motor center diagonal distance) racing frame. The motors used are T-Motor F40III KV2400 mated to 5040×3 propellers (Fig. 6a). The lower level controller and position hold is handled by ArduCopter 4.0.6 firmware running on the Holybro Kakute F7 flight controller mated to an optical flow sensor and TFMini LIDAR as altimeter source. All the higher level navigational commands are sent by the companion computer (NVIDIA Jetson TX2 running Linux for Tegra®) using RC-Override to the flight controller running in Loiter mode using MAVROS. The event camera used is a Samsung Gen-3 Dynamic Vision Sensor [13] with a resolution of 640×480 px. (Fig. 6c). and is mounted facing forward tilted down by 45° for the following experiment and facing down for the landing experiment. *All the computations and sensing are done on-board with no use of an external motion capture system.* Our neural network runs on a gutted Google Coral USB Accelerator with a custom heatsink attached to the TX2. The quadrotor take-off weight including the battery is 680 g and has a thrust to weight ratio of 5:1. Our network runs at 35Hz on the Coral accelerator (See Fig. 6b. Implementation details are given in Sec. V-D) and our planning and control algorithms run at 15 Hz on the TX2.

The larger quadrotor used in the following experiment is built on a S500 frame with DJI F2312 960KV motors mated to white colored 9450×2 propellers (Fig. 6d). Same avionics components are used as the smaller quadrotor.

The larger quadrotor used in the landing experiment is built on a S500 frame with T-Motor F80 Pro KV2500 motors mated to black colored 6040×3 propellers (Fig. 6a). Same avionics components are used as the smaller quadrotor and ArduCopter firmware holds the position in Loiter mode during experiments with all the sensor fusion, control and planning handled by the flight controller. *The area where the smaller quadrotor can land is of radius 135mm, which gives a tolerance of just 30mm on each side.*

B. Experimental Results And Observations

1) Quantitative Evaluation of EVPropNet

In the first case study we discuss quantitative evaluation results of our propeller detection results for varying resolution of propeller blade r_{px} (the bounding box size of the propeller would be $2r_{\text{px}}$ and is directly correlated with real-world propeller size r), number of blades N_{blades}, noise probability p_n, data miss probability p_b, different camera roll and pitch angles (ϕ and θ respectively). Formally, p_n denotes the probability with which a pixel can have error (equally likely to be either a positive or negative event) and p_b denotes the probability with which the pixel where the propeller data exists did not fire either due to a dead-pixel or camouflage with the background. We use the following metric to denote a successful detection of a propeller.

$$\text{Success} := {}^{\mathcal{G}\cap\mathcal{D}}/_{\mathcal{G}\cup\mathcal{D}} \geq 0.5; \mathcal{G} : \text{Ground Truth}, \mathcal{D} : \text{Detection} \quad (20)$$

Detection Rate DR is given by $\text{DR} = \mathbb{E}(\text{Success})$, where $\mathbb{E}$ is the expectation/averaging operator. The results are presented in Table II. When not specified, the values for the parameters are given as follows: $r_{\text{px}} = \{20, 30, 40, 50, 60\}$, $N_{\text{px}} = \{2, 3, 4, 5, 6\}$, RPM = {5K,10K,20K,30K,40K}, $p_n = \{0, 0.01, 0.02\}$, $p_b = \{0, 0.15, 0.3, 0.45, 0.6\}$, $\phi = \{0°, 10°, 20°, 30°, 60°\}$ and $\theta = \{0°, 10°, 20°, 30°, 60°\}$.

We see from Table IIa that DR increases with propeller size and then decreases, this is because as the amount of data increases, the results improve and but when the propeller is large ($r_{\text{px}} = 60$), we observe an increase in false detections near the edges of the propeller blades, dropping the DR slightly (Table IIa). A similar trend is observed with N_{blades}

Table II
DETECTION RATE (%) ↑ OF *EVPropNet* FOR VARIATION IN PARAMETERS.

(a) r_{px} (px.) for $\phi = \theta = 0°$

20	30	40	50	60
78.9	90.4	94.4	97.6	93.9

(b) N_{blades} for $\phi = \theta = 0°$

2	3	4	5	6
77.9	94.1	96.3	92.8	94.1

(c) RPM (min^{-1}) for $\phi = \theta = 0°$

5K	10K	20K	30K	40K
71.5	91.7	97.1	98.1	96.8

(d) p_n for $\phi = \theta = 0°$

0	0.01	0.02
92.6	91.4	89.1

(e) p_b for $\phi = \theta = 0°$

0	0.15	0.3	0.45	0.6
97.3	94.1	95.5	88.8	79.5

(f) ϕ (°) for $p_n = p_b = 0$

0	10	20	30	60
97.6	94.7	94.1	94.1	89.1

(g) θ (°) for $p_n = p_b = 0$

0	10	20	30	60
97.6	96.5	93.4	93.6	86.7

Table III
DETECTION RATE (%) ↑ OF APRILTAGS 3 FOR AMOUNT OF TAG BLOCKED.

p_b				
0	0.15	0.3	0.45	0.6
100	91.5	73.3	40.5	4.0

and RPM with DR peaking for a 4 bladed propeller and at 10K RPM (Tables IIb and IIc). We also observe that with increase in p_n (Table IId), the detection results are not affected significantly highlighting the robustness of our network. Even when 60% of the propeller is camouflaged with the busy background, we obtain a DR of above 79% (Table IIe) with the DR decreasing with increase in camouflage amount as expected. From Tables IIf and IIg, we also observe that even with camera angles (ϕ and θ) = 60°, we obtain a DR of above 85%. Finally, we obtain an overall DR of 85.1% for variations in all parameters and 90.9% when no data is corrupted.

Also, if we define success for drone detection as detecting atleast η propellers of the drone, we obtain the drone detection rate DR_D as follows $DR_D = (1 - (1 - DR)^{\eta})$, where DR is the detection rate of a single propeller. For example, we would obtain a drone detection rate DR_D of 97.7%, 99.6% and 99.9% for a quadrotor, hexacopter and octocopter respectively even when only 50% of the propellers are detected.

2) Quantitative Evaluation of April Tags 3

In the second case study, we evaluate how a custom designed passive fiducial marker would perform the task of detecting a drone (note that this is only applicable to a collaborative drone). In particular, we evaluate one of the most ubiquitous and robust passive fiducial markers April Tag 3 [21] (36h11 family) inspired from [40]. The parameters are the same as the first case study. From Table III, we observe that when the data is not missing (occluded or not correctly exposed), the April Tag detects the tags with an impressive DR (tag ID correctness is not considered) of 100%, but the accuracy falls significantly to 61.9% when data is missing (which is common in real-world due to high dynamic range scenarios and motion blur). It is also important to note the following reasons when a drone detection based on event camera based propeller detection will be better than a passive fiducial marker based detection.

- Detection of a non-collaborative drone for reconnaissance purposes
- High dynamic range and adverse lighting scenarios including fast movement
- Area occupied by non-occluded propellers is generally $\gg$ area occupied by the fiducial marker in the center (Refer to Sec. V-C for a detailed analysis)
- When a major part of the fiducial marker would be generally occluded

3) Quantitative Evaluation of Appearance based drone detectors

In the third case study, we compare drone detection using classical appearance based detectors from [15]. We see that the Haar Cascade detectors have a DR of 55.2% and the MobileNet deep learning based detector has a detection rate of 69.4% which are far lower than those of the fiducial detector and our propeller detection method.

4) Performance Measure on different compute platforms

In the fourth case study, we present speed and timing results for *EVPropNet* on various commonly used computational platforms. We refer the readers to [36] for a detailed description of the compute modules used in this case study. *EVPropNet* has 2.7M parameters, a model size of ∼10MB and utilizes 17GOPs for a single forward pass. We can see from Table IV that running *EVPropNet* to detect a drone by detecting propellers on the Google Coral Accelerator attached to the NVIDIA Jetson TX2 has the best speed and detection performance per unit power.

5) Qualitative Evaluation on different real-world propellers

In the final case study, we present qualitative results of *EVPropNet* on different lighting scenarios, propeller sizes, propeller and background colors, N_{blades}, r and angles. Fig. 7 shows the qualitative results where the description of the scene is given in Table V. Notice how *EVPropNet* can handle different real-world variations along with high dynamic range (Fig. 7c, even a high-end DSLR cannot capture both shadows and highlights with it's 32dB of dynamic range but the event camera with it's 80dB can handle such a scene with ease), low contrast (Fig. 7d) and low light with average intensity of 24lx (Fig. 7g).

6) Quantitative Evaluation of applications

We now present the results for both our applications and we call them experiment 1 for tracking and following and experiment 2 for landing. We define success as being able

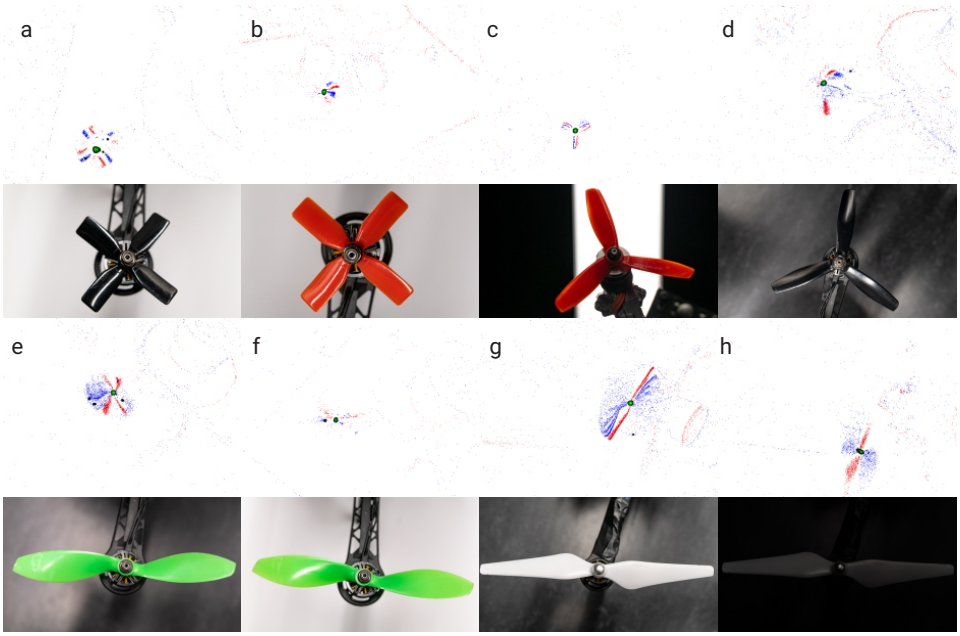

Figure 7. Top rows: Input event frame $\mathcal{E}$ where red and blue colors show positive and negative events respectively. Green color indicates *EVPropNet* prediction with the color saturation indicating confidence. Bottom rows: reference images of the propeller taken with a Nikon D850 DSLR (32dB dynamic range). Scenarios (a) to (h) are explained in Table V.

Table IV
PERFORMANCE METRICS ON DIFFERENT COMPUTE MODULES.

Method	(Ours) *EVPropNet* (Ours)					AprilTags 3 (36h11)	AprilTags 3 (16h5)
Computing Platform	PC (i9)	PC (TitanXp)	TX2†	NCS2*	Coral*	TX2	TX2
Speed ↑ (Frames per second)	8.6	133.4	10.5	4.5	35.2	7.0	41.3
Weight (g) ↓	–	–	130	138	136	130	130
Peak Power (W) ↓	250	250	15	17	17	15	15
Speed/Unit ↑ Power (FPS/W)	0.03	0.53	0.7	0.27	2.07	0.47	2.75
Detection Rate (%) ↑	85.1	85.1	85.1	83.4	81.9	61.9	53.4
Speed×DR/Unit ↑ Power (FPS%/W)	2.55	45.10	59.57	22.52	**169.53**	29.09	146.85

† Active heatsink removed. * Attached to TX2, outer casing removed and custom heatsink.

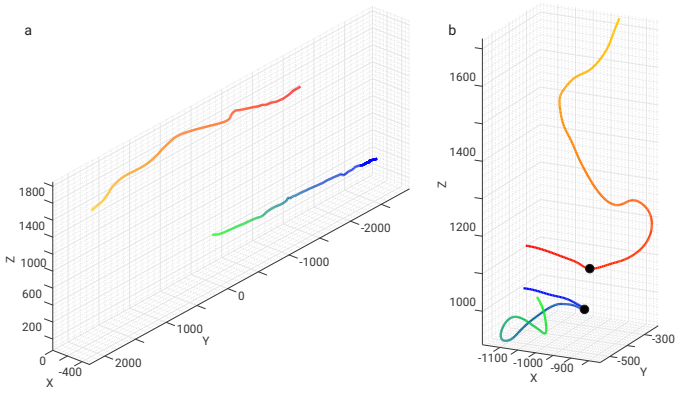

Figure 8. Vicon estimates for the trajectories of the smaller and larger quadrotor in the application experiments shown in Fig. 1. (a) Tracking and following, (b) Mid-air landing. Time progression is shown from yellow to red for the smaller quadrotor and and green to blue for the bigger quadrotor. The black dots in (b) show the moment in time where the touchdown occured.

detect the quadrotor and to not completely loosing track for experiment 1, and for the quadrotor to be able to detect the other quadrotor and land on it successfully without collision

Table V
DIFFERENT PROPELLER CONFIGURATIONS USED FOR QUALITATIVE EVALUATION IN FIG. 7.

Scenario	Ref. Fig.	Prop. Color	Background Color	Prop. Radius (mm)	Background Light Intensity (lx)	Propeller Area / Motor Area
(a)	7a	Black	White	50.8	240	2.3
(b)	7b	Red	White	50.8	240	2.3
(c)*	7c	Red	White and Black	63.5	564 and 2	3.6
(d)	7d	Black	Black	76.2	240	5.2
(e)	7e	Green	Black	88.9	240	7.1
(f)	7f	Green	White	88.9	240	7.1
(g)	7g	White	Black	119.4	24	12.8

* Case (c)'s light intensity shows High Dynamic Range scenario with illumination of the light part being 564lx and dark part being 2lx (See Fig. 7c).

for experiment 2. We average our results over 50 trails for each experiment and obtain a success rate of 92% for experiment 1 and 90% for experiment 2 (Vicon estimates for the trial shown in Fig. 1 are shown in corresponding sub-figures of Fig. 8). Commonly, the failure cases in experiment 1 happen when the larger quadrotor has a huge jerk that it moves outside the field of view of the camera. The failure cases in experiment 2 happen due to the aerodynamic interference between the two quadrotors which makes the bottom quadrotor drift at the last

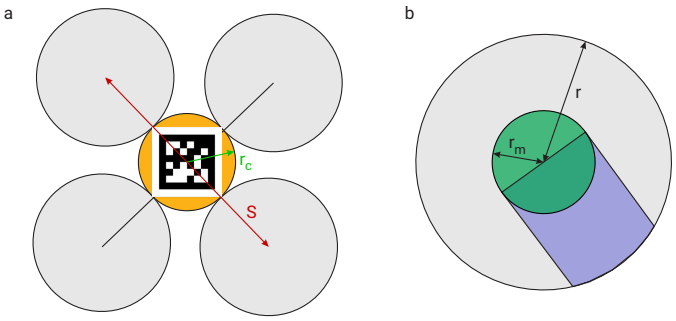

Figure 9. (a) Simplified model of a quadrotor used to calculate area ratios of the propellers to that of the biggest square fiducial marker that can be fit in the center without obstruction, (b) Simplified arm and motor projection to compute amount of propeller occluded from generating events – gray areas show where the propeller is visible and generates events, green area is occluded by the motor and blue area is occluded by the arm.

Table VI
$\mathcal{A}_{\text{ratio}}$ FOR SOME COMMON COMMERCIAL DRONES.

Name	S (mm)	N_{prop}	r (mm)	r_m (mm)	$\mathcal{A}_{\text{ratio}}$
DJI Phantom 4	350	4	119.4	12	109.8
QAV 210 X	210	4	63.5	14.0	51.2
DJI Inspire 2	603	4	190	18.5	69.2

moment.

C. Analysis

We present analysis of three questions: 1. What is the ratio of visible area of the propellers to that of the largest square fiducial marker in the center? (Fig. 9) 2. How does DR vary with focal length f, real-world propeller radius r and camera angle ϕ (angle around X axis)? (Fig. 2d). 3. What makes *EVPropNet* generalize to the real-world without any fine-tuning or re-training?

To analyse the answer to the first question, we present a simplified geometric model of a multirotor (quadrotor shown in Fig. 9a) where we are given a constraint on the drone's size S (diagonal motor to motor length) and number of propellers on the drone N_{prop}. Let us say that the largest fiducial marker that can fit in the center of the drone is inscribed in the circle of radius r_c. Also, we assume that the propeller does not generate events in the area in which is occluded by the arm and the motor. The motor radius is given by r_m and the arm width is given by $2r_m$ (Fig. 9b). The area of one non-occluded propeller $\mathcal{A}_{\text{prop}}$ (gray highlighted area in Fig. 9b) is given by

$$\mathcal{A}_{\text{prop}} = r^2\left(\pi - \frac{\gamma}{2}\right) - \frac{\pi r_m^2}{2} - r_m r\cos\left(\frac{\gamma}{2}\right); \gamma = 2\sin^{-1}\left(\frac{r_m}{r}\right) \tag{21}$$

Hence, the ratio for the area of the largest visible fiducial marker to that of a N_{prop} propeller drone will be

$$\mathcal{A}_{\text{ratio}} = \frac{4N_{\text{prop}}\left(r^2(2\pi-\gamma) - \pi r_m^2 - 2r_m r\cos\left(\frac{\gamma}{2}\right)\right)}{(S-2r)^2} \tag{22}$$

The value of $\mathcal{A}_{\text{ratio}}$ for some common commercially available drones are given in Table VI (Recall, N_{prop} is the number of propellers on the drone, r is the propeller radius, r_m is the motor radius, γ is defined in Eq. 21 and S is the drone's diagonal motor to motor length). We clearly see that the probability of observing at-least one propeller (directly related to $\mathcal{A}_{\text{ratio}}$) is much higher than that of observing a fiducial marker in the middle, thereby reinforcing the motivation of our approach.

For the analysis of the second question, refer to Fig. 10. We see that the DR of the propeller increases with an increase in real world propeller radius r until it reaches a maximum and then decreases (Fig. 10a). This trend is observed since smaller propellers (small r) generate a small number of events leading to a low DR and increases with increase in number of events (directly correlated with r). However, with larger propellers the DR decreases as the number of false detections increase near the tip of the propeller. With a larger focal length (larger f), the curvature of the curve is larger since the relative projection area change (on the image plane) is more drastic. We can also observe a similar trend in Fig. 10b with change in angle ϕ (angle around X^C axis in Fig. 2d). Notice that the change in focal length affects the accuracy more significantly than the change in angle. Note that pitch θ has the similar effect to that of the roll ϕ.

Finally, we speculate why our *EVPropNet* generalizes to the real-world without any fine-tuning or re-training for different propellers.

- The data's visual quality from simulation is similar to those obtained from recently developed event cameras both in-terms of noise and data-rate. (This is not simple with data from classical cameras due to the lack of photo-realism.)
- The errors in simulation (as compared to the real-world) are lower when the integration time for creation of event frames are smaller (around 20ms maximum) as demonstrated by [26].

D. Implementation Considerations

To speed-up the computation of our network when deployed on an aerial robot, we quantize our network to `Int8` and compile our network using `EdgeTPU` optimizations for deployment on the Google Coral USB Accelerator. To enable smooth compilation and high accuracy retention, we make our inputs take only valid `Int8` values as given below

$$\mathcal{I}_{\texttt{EdgeTPU}} = \text{clamp}\left(\mathcal{I} \times 255 + 127 | 0, 255\right) \tag{23}$$

$$\text{clamp}\left(x|a,b\right) := \max\left(b, \min\left(x, a\right)\right) \tag{24}$$

The labels $\hat{p}$ are modified as $\hat{p}_{\texttt{EdgeTPU}} = \lfloor \hat{p} \times 255 - 0.5 \rfloor$ and take integer values in $[0, 255]$.

Finally, when using an event camera with a high resolution at a high temporal sampling rate, the bottleneck of the system is the transfer speed between the event sensor and the compute module which are dictated by the combined throughput of processor, cache, transfer speeds of the primary and secondary memory. Such a bottleneck can cause data loss and data lag in the buffer. We mitigate this issue by using the NVIDIA TX2 which has a throughput of $\sim$440MBs^{-1}.

VI. Conclusions

We presented a method to detect unmarked drones (multi-rotors) by detecting a ubiquitous part of their design

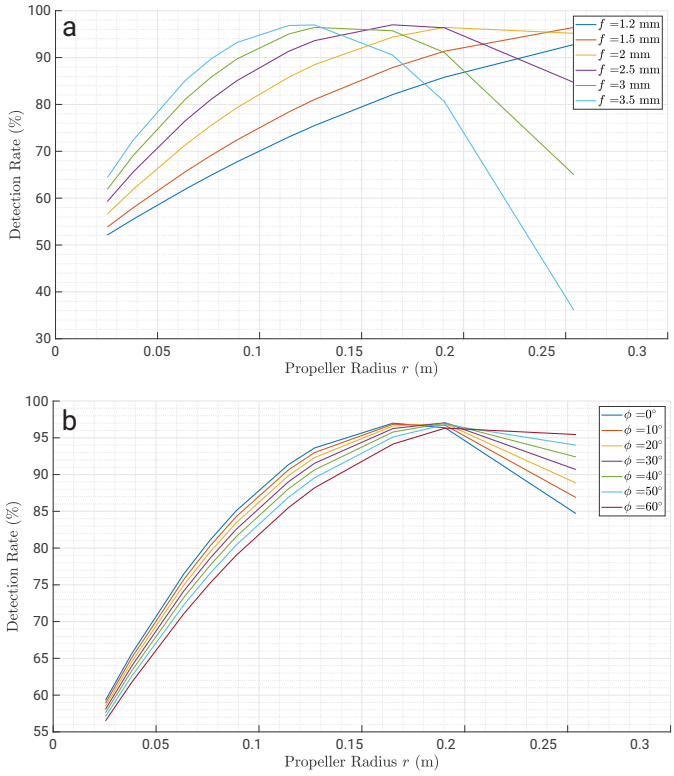

Figure 10. Variation of Detection Rate with variation in real-world propeller radius r for different (a) Focal lengths f with $\phi = 0°$, and (b) Camera Roll ϕ with $f = 2.5$mm.

– the propellers. To enable detection of the propellers, we utilize the following fact: propellers rotate at high-speed and hence are generally the fastest moving parts on an image. We model the geometry of the propeller and use it to simulate the data from an event camera whose qualities of high temporal resolution, low latency and high dynamic range make it perfectly suited for detecting propellers. We then train our *EVPropNet* deep network on this simulated data which generalizes directly to the real-world without any fine-tuning or re-training. We present two applications of detecting propellers on an unmarked drone: (a) tracking and following an unmarked drone and (b) landing on a near-hover drone. As a parting thought, an active zoom camera would increase the distance range from where the drones could be detected and would make our method a viable for deployment in the wild.

Acknowledgement

The support of the National Science Foundation under grants BCS 1824198 and OISE 2020624, the support of the Office of Naval Research under grant award N00014-17-1-2622, the Northrop Grumman Corporation and the Brin Family foundation are gratefully acknowledged. We also would like to thank Samsung for providing us with the event-based vision sensor used in this research.

References

[1] Teodor Tomic et al. Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue. *IEEE robotics & automation magazine*, 19(3):46–56, 2012.

[2] Kimberly McGuire, Mario Coppola, Christophe De Wagter, and Guido de Croon. Towards autonomous navigation of multiple pocket-drones in real-world environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 244–249. IEEE, 2017.

[3] KN McGuire, Christophe De Wagter, Karl Tuyls, HJ Kappen, and Guido CHE de Croon. Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment. *Science Robotics*, 4(35), 2019.

[4] Nitin J Sanket, Chahat Deep Singh, Varun Asthana, Cornelia Fermüller, and Yiannis Aloimonos. Morpheyes: Variable baseline stereo for quadrotor navigation. *arXiv preprint arXiv:2011.03077*, 2020.

[5] Tolga Özaslan et al. Inspection of penstocks and featureless tunnel-like environments using micro UAVs. In *Field and Service Robotics*, pages 123–136. Springer, 2015.

[6] Friedrich Fraundorfer, Lionel Heng, Dominik Honegger, Gim Hee Lee, Lorenz Meier, Petri Tanskanen, and Marc Pollefeys. Vision-based autonomous mapping and exploration using a quadrotor mav. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4557–4564. IEEE, 2012.

[7] Nathan Michael et al. Collaborative mapping of an earthquake-damaged building via ground and aerial robots. *Journal of Field Robotics*, 29(5):832–841, 2012.

[8] Nitin J Sanket, Chahat Deep Singh, Kanishka Ganguly, Cornelia Fermüller, and Yiannis Aloimonos. Gapflyt: Active vision based minimalist structure-less gap detection for quadrotor flight. *IEEE Robotics and Automation Letters*, 3(4):2799–2806, 2018.

[9] Daniel Mellinger, Michael Shomin, Nathan Michael, and Vijay Kumar. *Cooperative Grasping and Transport Using Multiple Quadrotors*, pages 545–558. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[10] Zoran Valentak. Drone market share analysis, 2018.

[11] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jörg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *CoRR*, abs/1904.08405, 2019.

[12] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128× 128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, 2008.

[13] B. Son, Y. Suh, S. Kim, H. Jung, J. Kim, C. Shin, K. Park, K. Lee, J. Park, J. Woo, Y. Roh, H. Lee, Y. Wang, I. Ovsiannikov, and H. Ryu. 4.1 a 640×480 dynamic vision sensor with a 9µm pixel and 300meps address-event representation. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 66–67, 2017.

[14] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[15] Maciej Ł Pawełczyk and Marek Wojtyra. Real world object detection dataset for quadcopter unmanned aerial vehicle detection. *IEEE Access*, 8:174394–174409, 2020.

[16] Eren Unlu, Emmanuel Zenou, Nicolas Riviere, and Paul-Edouard Dupouy. Deep learning-based strategies for the detection and tracking of drones using several cameras. *IPSJ Transactions on Computer Vision and Applications*, 11(1):1–13, 2019.

[17] Fabian Schilling, Fabrizio Schiano, and Dario Floreano. Vision-based drone flocking in outdoor environments. *IEEE Robotics and Automation Letters*, 6(2):2954–2961, 2021.

[18] Viktor Walter, Nicolas Staub, Antonio Franchi, and Martin Saska. Uvdar system for visual relative localization with application to leader–follower formations of multirotor uavs. *IEEE Robotics and Automation Letters*, 4(3):2637–2644, 2019.

[19] Luis A. Mateos. Apriltags 3d: Dynamic fiducial markers for robust pose estimation in highly reflective environments and indirect communication in swarm robotics, 2020.

[20] Guanrui Li, Bruno Gabrich, David Saldana, Jnaneshwar Das, Vijay Kumar, and Mark Yim. Modquad-vi: A vision-based self-assembling modular quadrotor. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 346–352. IEEE, 2019.

[21] Maximilian Krogius, Acshi Haggenmiller, and Edwin Olson. Flexible layouts for fiducial tags. In *IROS*, pages 1898–1903, 2019.

[22] Lilian Calvet, Pierre Gurdjos, Carsten Griwodz, and Simone Gasparini. Detection and Accurate Localization of Circular Fiducials under Highly Challenging Conditions. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 562 – 570, Las Vegas, United States, June 2016.

[23] Anton Mitrokhin, Cornelia Fermüller, Chethan Parameshwara, and Yiannis Aloimonos. Event-based moving object detection and tracking. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[24] Timo Stoffregen, Guillermo Gallego, Tom Drummond, Lindsay Kleeman, and Davide Scaramuzza. Event-based motion segmentation by motion compensation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7244–7253, 2019.

[25] Chethan M Parameshwara, Nitin J Sanket, Chahat Deep Singh, Cornelia Fermüller, and Yiannis Aloimonos. 0-mms: Zero-shot multi-motion segmentation with a monocular event camera.

[26] Nitin J Sanket, Chethan M Parameshwara, Chahat Deep Singh, Ashwin V Kuruttukulam, Cornelia Fermüller, Davide Scaramuzza, and Yiannis Aloimonos. Evdodgenet: Deep dynamic obstacle dodging with event cameras. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10651–10657. IEEE, 2020.

[27] Davide Falanga, Kevin Kleber, and Davide Scaramuzza. Dynamic obstacle avoidance for quadrotors with event cameras. *Science Robotics*, 5(40), 2020.

[28] John Carlton. *Marine propellers and propulsion*. Butterworth-Heinemann, 2018.

[29] Wenchao Ding, Wenliang Gao, Kaixuan Wang, and Shaojie Shen. An efficient b-spline-based kinodynamic replanning framework for quadrotors. *IEEE Transactions on Robotics*, 35(6):1287–1306, 2019.

[30] Eric W. Weisstein. B-spline. From MathWorld—A Wolfram Web Resource.

[31] Kaihuai Qin. General matrix representations for b-splines. *The Visual Computer*, 16(3-4):177–186, 2000.

[32] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

[33] Henri Rebecq, Daniel Gehrig, and Davide Scaramuzza. ESIM: an open event camera simulator. *Conf. on Robotics Learning (CoRL)*, October 2018.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[35] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018.

[36] Nitin J Sanket, Chahat Deep Singh, Cornelia Fermüller, and Yiannis Aloimonos. Prgflow: Benchmarking swap-aware unified deep visual inertial odometry. *arXiv preprint arXiv:2006.06753*, 2020.

[37] Philipp Foehn, Dario Brescianini, Elia Kaufmann, Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. AlphaPilot: Autonomous Drone Racing. In *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020.

[38] Karan P Jain and Mark W Mueller. Flying batteries: In-flight battery switching to increase multirotor flight time. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3510–3516. IEEE, 2020.

[39] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[40] Bernd Pfrommer, Nitin Sanket, Kostas Daniilidis, and Jonas Cleveland. Penncosyvio: A challenging visual inertial odometry benchmark. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3847–3854. IEEE, 2017.