Temporal Siamese Networks for Clutter Mitigation Applied to Vision-Based Quadcopter Formation Control

James Dunn[1] and Roberto Tron[2]

***Abstract*— Convolutional neural networks applied to video streams often suffer from short-lived misclassifications or false alarms from clutter and noise. We introduce a novel network training method based on the Siamese Networks technique that mitigates short-lived false alarms in an Hourglass CNN that segments out quadcopters in a live video stream. To demonstrate this method in a real-world application in real-time, we implement it as part of a quadcopter tracker for vision-based formation control.**

Quadcopter drone formation control is an important capability for fields like area surveillance, search and rescue, agriculture, and reconnaissance. Of particular interest is formation control in environments where wireless communications and/or GPS may be either denied or not sufficiently accurate for the desired application. Using vision to guide the quadcopters addresses these situations, but computer vision algorithms are often computationally expensive and suffer from high false clutter detection rates. Our novel Siamese networks-based clutter mitigation technique is a good way to mitigate this clutter without added computational complexity at run-time.

We run our real-time implementation on a single-board computer (ODROID XU4) with a standard webcam mounted to a quadcopter drone. Flight tests in a motion capture volume demonstrate successful formation control with two quadcopters in a leader-follower setup.

I. INTRODUCTION

A. MOTIVATION

Quadcopter drones are ubiquitous in today's market. They are used for everything from filmmaking to search and rescue. Their low cost and small size makes formations of numerous quadcopters more feasible and also allows amateur hobbyists to fly them. Unfortunately this also means they have limited onboard processing capability and limited onboard sensing capability.

Standard visual cameras are low-size, low-weight, and low-cost sensors, which makes them ideal for mounting onto quadcopter drones. Their video feeds contain a wealth of information that can be used for object recognition and navigation, much like how humans and other animals use sight.

While video feeds are rich sources of information, processing the video feed from a visual camera is processor-intensive and error prone due to the volume of the datastream, short latency requirements of quadcopter drone feedback control ($\leq$ 100 milliseconds), and clutter-rich environments of interest. Processing the video feed onboard the quadcopters themselves grants independence from a ground base-station or a human operator, but presents a limitation on the available processing power.

Standard approaches to quadcopter formation control rely on either offboard processing, GPS, or wireless communication between the quadcopters [1] [2]. In many environments, wireless communications and/or GPS may be denied or not sufficiently accurate for the desired application. This is common indoors and in military scenarios due to electronic interference or jamming. A handful of recent studies use a video feed to guide the formation control, but they rely on fiducial markers to aid detection and localization [3] [4] [5] [6] [7] [8] [9].

Our implementation does not rely on any of these, following the lead of [10] and [11]. We use only the video feed from the quadcopters' onboard cameras, and process it with an onboard ODROID XU4. We use a trained hourglass CNN to generate a quadcopter/non-quadcopter segmentation for each frame. We use the segmentation to localize the imaged quadcopter, eliminating the need for fiducial markers. This enables "non-cooperative" formation control. For example, trailing or chasing a rogue quadcopter, or intentionally colliding with a quadcopter that poses a threat.

High false detection rates plague visual quadcopter detection and localization, making stable flight difficult. In our implementation, the naïve trained hourglass CNN's segmentations suffer from false clutter detections that bias the resulting quadcopter navigation solution.

B. SOLUTION

To address the poor localization resulting from high false detection rates, we implement a novel method of mitigating displaced false clutter detections *during network training* that is based on the Siamese networks technique. Specifically, during training we add a loss penalty when consecutive frames have dissimilar segmentations. The network consequently learns to avoid promoting image features that tend to be short lived.

The resulting trained network has exactly the same size and shape as the original, but its trained parameters better avoid false clutter detections. This minimizes the need for computationally-expensive post-processing clutter mitigation algorithms. This "temporal Siamese clutter mitigation" is not limited to quadcopter segmentation or even Hourglass CNNs

*DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the United States Air Force under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

[1]James Dunn is a graduate student in Electrical and Computer Engineering, Boston University, 8 St Mary's St, Boston, MA 02215, USA jkdunn@bu.edu

[2]Roberto Tron is an Assistant Professor of Mechanical and Systems Engineering, 8 St Mary's St, Boston, MA 02215, USA tron@bu.edu

- any application where the network output is expected to be smooth in time can benefit from this method.

Using a binary quadcopter/non-quadcopter segmentation requires assuming a physical size of the imaged quadcopter. This is not a problem for a formation with friendly quadcopters, but for non-friendly quadcopters we have to make an educated guess. A quadcopter "make and model" classification algorithm could be run in parallel to our segmentation algorithm as a way to determine physical size. We leave the implementation of such an algorithm to future studies.

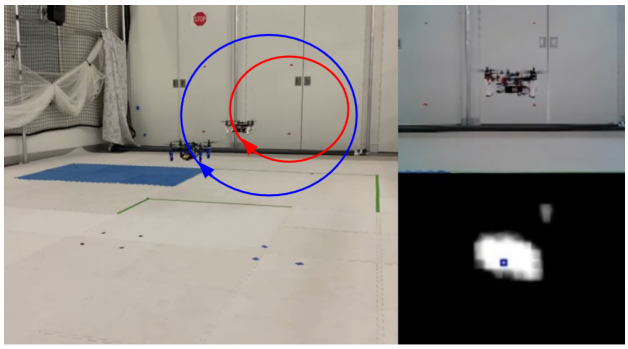

Fig. 1. Two quadcopters with the associated video feed and heatmap from the follower

II. BACKGROUND

A. Hourglass and Siamese Neural Networks

Hourglass CNNs, also known as U-networks, add *upconvolution* layers after conventional convolution and pooling layers to generate full-resolution images as the network output. Upconvolution layers learn blocks of weights that attempt to accurately up-sample the data in the network to a finer resolution. Hourglass CNNs also employ "skip connections", which copy the output of earlier layers into the latter half of the network to aid in resolution enhancement and prevent the vanishing gradients problem during backpropagation.

Hourglass CNNs are primarily used for image segmentation. They were first applied in the mid-1990's by [12] for text-vs-image segmentation. Since then, hourglass CNNs have been used in a variety of contexts, including camera pose estimation [13], human joint and pose localization [14] [15], and tumor segmentation in medical imaging [16]. Li et. al. [17] have a particularly relevant application of hourglass CNNs: they utilize a "Contextual Hourglass Network" to segment imagery taken from aerial platforms, but they don't use it for autonomous flight control and don't apply the Siamese-networks technique.

An common alternative to using an Hourglass CNN to generate a quadcopter/non-quadcopter segmentation is to implement a Haar Cascade-based network [18]. Haar classifiers are fast and preform well detecting specific objects, but only give a bounding-box rather than a full pixel-by-pixel segmentation. For our application, having an accurate representation of the quadcopter size gives us an accurate measure of range, which a Haar-based classifier would not provide. Using a segmentation also provides the framework for a more specific drone "make and model" type of classifier.

The "Siamese networks" technique refers to running two identical copies of the same neural network on different input data, then making a comparison between the outputs of the two networks. It was first developed by Bromley et. al. in 1994 [19] for the purpose of handwritten signature matching.

Siamese networks are commonly used for binary match/no match decision making between the two inputs, such as single-shot image recognition [20] and comparison of image patches [21]. Bertinetto et. al. [22] implement a particularly relevant Siamese network for tracking an object through a video given just the localization of the object in the first video frame. Other uses include calculating the similarity of sentences [23] and scene detection in broadcast videos [24].

We use the Siamese networks technique to compare the quadcopter likelihood heatmaps from temporally adjacent video frames during training. Specifically, we penalize the network for making the heatmaps from consecutive frames different from one another. The result is that the network learns to make heatmaps that are smoother in time, which leads to less chattering in the estimated location of the leader and thus less chatter in the movement of the follower. This requires no additional processing during flight - the benefits are embedded within the trained network. To our knowledge, training an Hourglass CNN with this Siamese networks technique has not been attempted before, and it is the novel contribution of this study.

B. Vision-Based Quadcopter Formation Control

Formation control - moving robotic agents around while maintaining their positions relative to one another - is a popular and useful research topic. Successful execution requires some way of determining the relative position and motion of the agents. This is usually done with wireless communication between the agents and/or GPS, but as noted earlier these can be unreliable.

Using vision to localize the quadcopters makes formation control robust to unreliable communication links. The standard way of making visual localization of the quadcopters easier is by attaching some sort of fiducial marker(s) to them. The authors of [3] execute a leader-follower formation on two quadcopter drones using bright colored infrared LEDs on the leader to enable visual pose estimation. Similarly, in [4] the authors use ultraviolet markers on the quadcopters to aid localization. In [5], the authors implement formation control with three quadcopter drones using onboard cameras. To ease visual localization, each quadcopter has a large, uniquely-colored fiducial disk mounted to it. The authors of [6] implement a vision-based method of quadcopter detection to feed a leader-follower formation using *Aruco* fiducial markers to ease visual localization. The authors of [7] and [8] implement MAV swarm stabilization in an outdoor environment without use of GPS or inter-MAV communication, and also use fiducial markers on the quadcopters to aid localization. In

[9] the authors use concentric circle markers hanging under each UAV.

A pair of recent studies, [10] and [11], use neural networks to remove this reliance on fiducial markers like we do. This makes it easier to implement with arbitrary quadcopters. Not relying on fiducial markers also makes our approach extensible to scenarios where the leader is non-cooperative, such as seek-and-destroy or covert reconnaissance missions.

In [10] the authors train via simulation and implement with hardware a vision-based *and fiducial-free* algorithm to guide a drone swarm. They run flight tests with two quadcopters, focusing on collision avoidance and cohesion rather than rigid formation control. The primary difference between their study and ours is their use of imitation learning to go directly from camera images to velocity commands as the output of the neural network. We use an Hourglass CNN's output heatmaps to explicitly localize the leader before applying velocity commands. Consequently, with our approach the quadcopters can move in a prescribed rigid formation that is determined *after* network training, rather than just remaining close to one another while avoiding collisions as is done in [10].

In [11] the authors demonstrate MAV formation control using an existing CNN that generates bounding boxes around the detected MAVs. Like our study, they use a single onboard camera, onboard processing, and a CNN in lieu of fiducial markers. In contrast with our study, their detection CNN generates bounding boxes. Our CNN generates pixel-by-pixel segmentations and takes advantage of the novel Siamese-network clutter mitigation strategy. This has the benefit of giving more accurate range estimations and improving performance against clutter. It also lays the groundwork for using the shape of the segmented quadcopter to aid in identification for formations with more than two quadcopters.

III. Methods

A. Data Collection

Training data was generated via greenscreen injection. Specifically, the quadcopter was held by hand in front of a greenscreen and moved around while a video was taken of it. The greenscreen and skin pixels from the hand and arm are removed in post-processing using their unique colors per [25] and [26]. The resulting quadcopter pixel mask is used to inject the quadcopter over a series of background videos. The pixel mask is also used as ground truth in network training, specifically as the pixel-by-pixel truth map that enables the loss calculation in Equation 2.

The primary performance analysis and flight tests used a network trained with background videos collected in the Boston University robotics laboratory. We also ran a secondary performance analysis, training the network with background videos in a diverse outdoor environment. The background videos include a number of challenging clutter objects, including other non-quadcopter robots.

We applied standard data augmentation on the injected quadcopter videos to expand the training set with representative images. We scale all images to 96 $\times$ 72 monochrome pixels with 8-bit depth. We end up with a total of 210,688 monochrome 96 $\times$ 72 pixel images to train our Siamese-Hourglass CNN as detailed in Section III-B.

B. Siamese Hourglass CNN

The heart of our quadcopter localization algorithm is an hourglass CNN that generates heatmaps: pixel-by-pixel quadcopter likelihood maps. We use a 4-layer hourglass network, following the general architecture of [13]. See Figure 2 for a diagram of the network architecture, and Table I for a layer-by-layer breakdown.

TABLE I

HOURGLASS NETWORK LAYER-BY-LAYER BREAKDOWN

layer	operation	weight shape	stride	output shape
Input	-	-	-	96×72
1	conv2d	$5 \times 5 \times 1 \times 32$	1×1	$96 \times 72 \times 32$
1	relu	-	-	$96 \times 72 \times 32$
1	maxpool	-	4×4	$24 \times 18 \times 32$
2	conv2d	$5 \times 5 \times 32 \times 64$	1×1	$24 \times 18 \times 64$
2	relu	-	-	$24 \times 18 \times 64$
2	maxpool	-	2×2	$12 \times 9 \times 64$
3	upconv2d	$3 \times 3 \times 32 \times 64$	2×2	$24 \times 18 \times 32$
3	relu	-	-	$24 \times 18 \times 32$
3	skip connection	-	-	$24 \times 18 \times 64$
4	upconv2d	$5 \times 5 \times 1 \times 64$	4×4	96×72
4	relu	-	-	96×72
Output	-	-	-	96×72

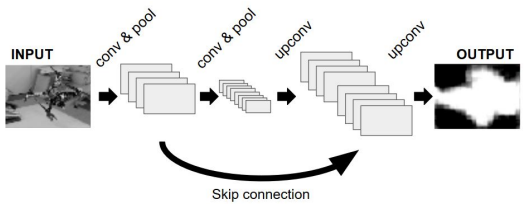

Fig. 2. Diagram of the hourglass neural network architecture, from input 96×72 raw image $\mathbf{F_t}$ to output 96×72 heatmap $\mathbf{H_t}$.

The primary reason for using the upsampling layers in the latter half of the network is to enable an accurate measurement of the area of the detected quadcopter and thus an estimate of the range to it. If we were to omit the upsampling layers - effectively using the CNN to directly generate a coarse estimate of bearing - we would not get an accurate range estimate.

Conversely, we choose to use a neural network *only* for segmentation. We leave the explicit localization and quadcopter velocity calculation to be done in post-processing. Neural networks can be designed to directly calculate quadcopter velocity commands from the video feed, as in [10]. Building the network in this way however means embedding a choice of formation shape into the trained network. Exiting the network after segmentation thus allows for more flexibility with a single trained network.

The time-sequence of heatmaps output from the hourglass CNN suffer from short lived blobs of false-target pixels. These false-target pixel blobs result in "chattering"

of the quadcopter localization, and consequently lead to non-smooth flight controls. To address this chattering, we use the Siamese networks technique in a novel way.

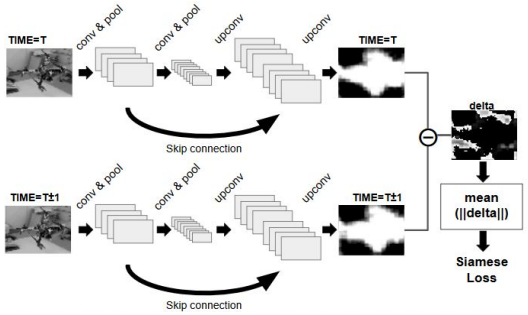

Fig. 3. Diagram of the Siamese hourglass neural network architecture

In broad terms, the "Siamese networks technique" involves running two identical copies of the same network simultaneously but with different input images, and then comparing the output of the two networks to benefit the desired application. In our specific implementation, we apply the Siamese networks technique on consecutive temporal frames of the training videos. We train the network output from consecutive frames to be similar by adding the magnitude of their difference to the final network loss term. We call the average of this difference the "Siamese loss". See Figure 3 for a diagram of the full Siamese hourglass network architecture used in this study.

Minimizing the Siamese loss teaches the network that input images with minimal differences should have similar resulting heatmaps. In particular, small motions of the target and background should produce minimal differences in the resulting heatmaps. This effect is similar to that of a temporal smoothing function like a Kalman filter, but it works to directly mitigate false clutter in the imagery rather than using multiple measurements to average it away. The Siamese technique tends to remove non-random and persistent clutter, while the Kalman filter averages away randomly distributed clutter and noise. The Siamese technique also runs on frames one-at-a-time, while the Kalman filter relies on combining multiple measurements.

The Siamese loss need-not be calculated between consecutive frames. For example, it can be calculated between any two frames a set time apart from one another. Through experimenting, we found that using frames up to approximately 0.12 seconds apart has comparable clutter mitigation performance to using consecutive frames (0.04 seconds apart in our implementation). We choose to use consecutive frames for simplicity.

C. Heatmap Loss Term

The Siamese-Hourglass CNN's objective function consists of two terms: a "heatmap loss" L_t^H that trains the network to make output heatmaps that match the associated truth masks from the greenscreen injections, and a "Siamese loss" L_t^S that trains the network to mitigate short-lived false alarm clusters in the heatmaps.

Equations 1 through 3 show the calculation of the heatmap loss term L_t^H for a single 96×72 pixel input image at time t, $\mathbf{F_t}$. We start with the output of the hourglass neural network described in Figure 2 and Table I, $\mathbf{H_t} = \mathbf{H_{CNN}}(\mathbf{F_t})$:

$$\mathbf{P_t} = \begin{cases} 1, & \text{where } \mathbf{H_t} \geq 1 \\ \mathbf{H_t}, & \text{otherwise} \end{cases} \tag{1}$$

$$\mathbf{M_t} = \begin{cases} 1, & \text{where pixel is "target"} \\ -w, & \text{where pixel is "background"} \end{cases} \tag{2}$$

$$L_t^H = -|\mathbf{P_t} \odot \mathbf{M_t}| \tag{3}$$

Where:

- We cap the heatmap at 1 in Equation 1 to prevent pixels with large heatmap values from dominating the backpropogation gradients
- We use pixel-by-pixel truth from the greenscreen injection to generate $\mathbf{M_t}$
- w is a weight (hyper-parameter) that penalizes false positives. Nominally 0.01.
- The $|\cdot|$ operator means "average over all pixels".
- The $\odot$ operator denotes the Hadamard (pixel-by-pixel) product

In short, a correctly identified quadcopter pixel decreases L_t^H by $\approx \frac{1}{N}$, and an incorrectly identified quadcopter pixel increases L_t^H by $\approx \frac{w}{N}$.

D. Siamese Loss Term

For each training image $\mathbf{F_t}$, we run the image taken immediately before or after and with the same applied data augmentations, $\mathbf{F_{t\pm1}}$, through the neural network to generate $\mathbf{P_{t\pm1}}$. We randomize the choice of prior frame $\mathbf{F_{t-1}}$ vs next frame $\mathbf{F_{t+1}}$ to avoid biasing the network to predict motion in any one direction. We compare the heatmap from the adjacent frame, $\mathbf{P_{t\pm1}}$, to the heatmap of the frame of interest, $\mathbf{P_t}$, to calculate a single Siamese loss term, L_t^S as follows:

$$L_t^S = |\,(\mathbf{P_t} - \mathbf{P_{t\pm1}})^2\,| \tag{4}$$

E. Objective Function and Training

The weighted sum of the heatmap loss and Siamese loss terms forms the final loss, L_t per:

$$L_t = L_t^H + w_s L_t^S \tag{5}$$

Where:

- w_s is a weight (a hyper-parameter), empirically selected to maximize the network's overall performance. Values for $w_s \in [0.01, 10.0]$ were tested. Larger w_s tends to minimize false clutter detections away from the quadcopter, and smaller w_s tends to increase sharpness at quadcopter edges. $w_s = 0.1$ shows the best performance (see figure 6), and we use $w_s = 0.1$ for all flight tests.

We train and analyze using standard 4-fold cross validation on the 210,688 post augmentation video frames. Care was taken to ensure that augmented versions of the same raw

input frame all stayed within the same fold to prevent cross-contamination of train and test sets.

We use Tensorflow's built-in ADAM optimizer [27] on the loss L_t from (5) to perform backpropogation and network weight training. We ran 194 training epochs over 60,000 iterations (512 images/batch, 158,016 training images/fold). Examination of the train and test loss vs number of epochs showed convergence.

F. Flight Controls

We upload the trained Siamese-Hourglass CNN onto the follower's ODROID XU4. The ODROID XU4 streams video frames, $\mathbf{F_t}$, from its attached USB video camera through the trained network in real time to generate heatmaps $\mathbf{H_t}$. Note that the real-time calculation of $\mathbf{H_t}$ does *not* depend on the prior or next frame $\mathbf{F_{t\pm1}}$, because the Siamese loss term is only used for training.

We apply a minimum threshold h_m to the heatmap $\mathbf{H_t}$ to mitigate spurious clutter and noise, and then calculate the centroid pixel (mathematical first moment) $\mathbf{c_t}$ as:

$$\mathbf{H_t^*} = \begin{cases} \mathbf{H_t}, & \text{where } \mathbf{H_t} \geq h_m \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

$$\mathbf{c_t} = \frac{\sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \mathbf{x} \mathbf{H_t^*}[\mathbf{x}]}{N} \tag{7}$$

Where:

- $\mathbf{x}$ is the 2-D pixel index $\binom{i}{j}$ in $\mathbf{H_t^*}$
- N is the number of pixels in the heatmap. Nominally $N_x \times N_y = 96 \times 72 = 6912$.

Likewise, we calculate the area of the thresholded heatmap as a fraction of the frame's area, $\mathbf{A_t}$:

$$\mathbf{H_t^+} = \begin{cases} 1, & \text{where } \mathbf{H_t} \geq h_m \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

$$\mathbf{A_t} = |\mathbf{H_t^+}| \tag{9}$$

We convert the centroid $\mathbf{c_t}$ and area $\mathbf{A_t}$ into a 3D position $\mathbf{s}_t$ of the leader using the known camera field of view (37°-by-28°) and an assumed size of the leader quadcopter. We run $\mathbf{s}_t$ through a standard Kalman filter to smooth out the position estimate and enable coasting of the Kalman state when the leader leaves the follower's field of view. The Kalman filter is computationally inexpensive and complements the network's embedded clutter mitigation well.

We then use the Kalman-filtered position of the leader $\mathbf{s}_t$ to determine the location that the follower needs to fly to in order to maintain the desired formation. We call this the "desired setpoint". We employ an attractive quadratic potential with a 16 cm diameter dead zone and maximum speed of 0.2 m/s to calculate the velocity commands that move the follower toward the desired setpoint.

The quadratic potential was chosen for simplicity - other controllers would also suffice and result in slightly different flight profiles. The size of the dead zone was chosen to minimize erratic motions during flight ("chattering").

The formation control applied in this study does not account for or require knowing the orientation of the lead quadcopter. The quadcopter detection algorithm does not provide a means to determine orientation.

IV. Flight Test Setup

For flight testing, we use a pair of functionally-identical custom-built quadcopters. The quadcopters measure 50cm × 50cm × 15cm (including propellers) with a mass of 1.1 kilograms. They are built from mostly commercial-off-the-shelf (COTS) parts, plus some 3D printed custom frame components.

Each quadcopter uses a 3DR Pixhawk Mini flight controller and a single-board ODROID XU4 computer running Ubuntu 16.04 with ROS Kinetic. A standard USB color webcam is mounted to the front of the follower and captures low-resolution monochrome video. The leader flies a scripted path. Software implementation is in Python using the MAVROS, OpenCV, and Tensorflow libraries.

Flight tests are executed in the Boston University Robotics Laboratory, which is outfitted with a 44-camera Optitrack motion capture system for real-time tracking of the quadcopters for analysis and flight safety.

V. Experimental Results

A. Siamese-Hourglass CNN Performance

We start with some example images run through the hourglass CNN to show the segmentation algorithm in action. Figure 4 shows some examples of input images $\mathbf{F_t}$ with their output heatmaps $\mathbf{H_t}$ and a color-coded map of the pixel-by-pixel predictions vs truth.

We use the following standard definitions:

- True positives: Pixels classified as part of a quadcopter that are part of a quadcopter
- False positives: Pixels classified as part of a quadcopter that are actually background
- False negatives: Pixels classified as background that are actually part of a quadcopter
- True negatives: Pixels classified as background that are background

Figure 5 shows four example images next to their respective pixel-by-pixel prediction vs truth maps, with and without using the Siamese technique when training the CNN. Notice how the Siamese technique mitigates the misplaced red blobs of false positives. This is the primary benefit of including the Siamese loss during training.

We calculate Siamese Hourglass CNN performance metrics by aggregating the predictions for every pixel in the data set and using 4-fold cross validation. Figure 6 shows precision/recall (P/R) curves for the trained Hourglass CNN with and without applying the Siamese loss term L_t^s during training. P/R curves are plotted using different values for the Siamese loss weight w_s. Values for the Siamese loss weight $w_s \in [0.01, 10.0]$ were tested, and we plot P/R curves for $w_s = [0.05, 0.1, 0.2]$.

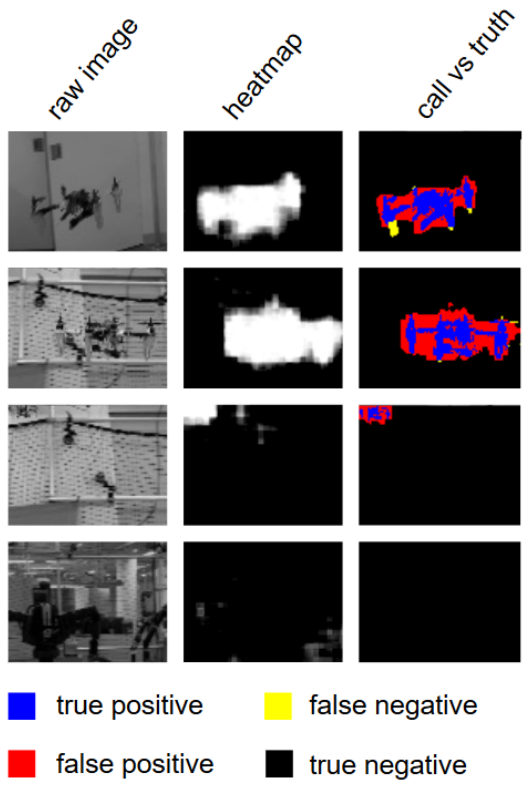

Fig. 4. Example images run through the hourglass CNN and the pixel-by-pixel calls vs truth

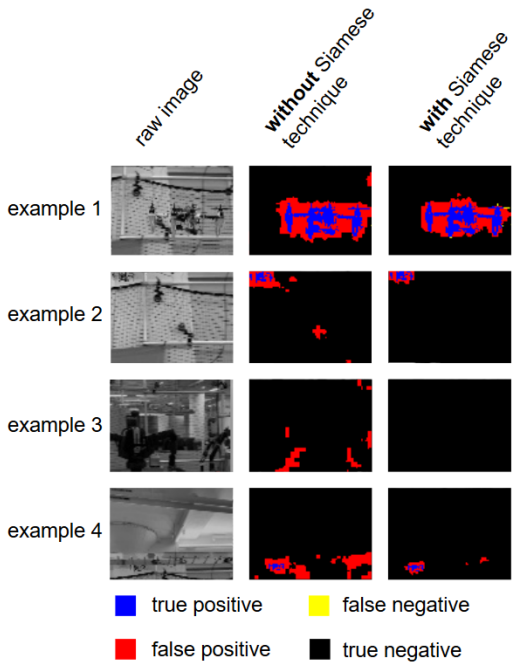

Fig. 5. Example frames showing the benefit of using the Siamese technique

Including the Siamese loss during training clearly improves detection performance. Values for w_s in the neighborhood of 0.1 all show similar performance, with $w_s = 0.1$ performing slightly better near the operating point of interest. The blue diamond in Figure 6 is the operating point on the $w_s = 0.1$ curve that we use for flight tests.

As evidenced by our choice of operating point on the P/R curve, we leverage the Siamese networks technique to trade a few true positives for a large reduction in false positives. False positives are detrimental to our flight control, especially when they are located far from the actual position of the leader. The small loss to recall is of little consequence for well-resolved quadcopters.

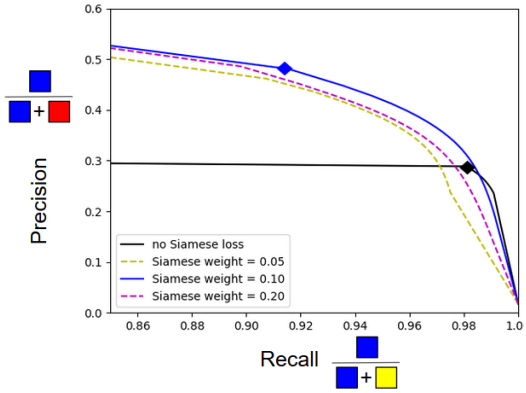

Fig. 6. P/R curves with Siamese Loss term L_i^s with different Siamese weights w_s. The blue diamond marks the operating point (determines the heatmap threshold $h_m = 0.85$) used in flight tests.

To check that the Siamese hourglass segmentation extends to alternate environments, we trained and tested with a background video taken outdoors with a mix of trees, cars, and buildings. We used greenscreen quadcopter injections and 4-fold cross validation as we did with the laboratory background videos. Figure 7 shows some example frames and figure 8 shows the corresponding P/R curves with and without applying the Siamese technique during training.

B. Flight Test Results

We show the results of two flight tests, one where the quadcopters trace out a vertical circle and one where they trace out a pyramid-esque shape. Videos for these flights and others are included in the supplementary files for this article. Figure 9 shows plots of the track error (distance between follower's actual position and its desired setpoint) and 3D positions for one cycle through each motion.

Across a larger set of flight tests, our average track error was 30 cm. This is the superposition of the 16 cm dead zone in the quadratic potential, the 18 cm average error in the follower's ability to measure the position of the leader, and similar error in the Pixhawk flight controller's ability to accurately fly the commanded trajectory. The largest track errors occur when the leader executes a high-acceleration maneuver. Similarly, the smallest track errors occur during near constant-velocity flight.

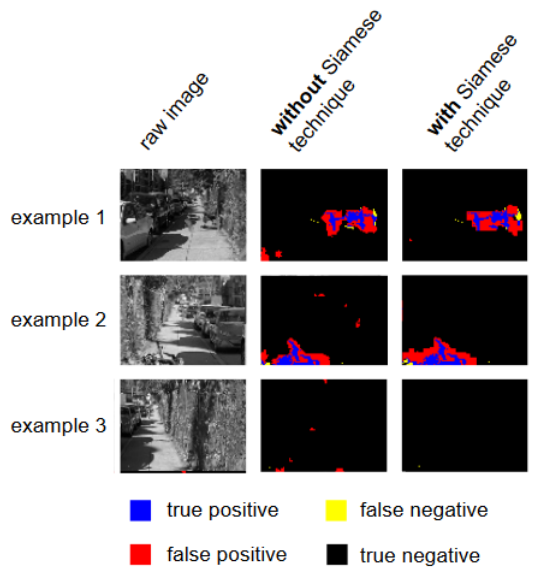

Fig. 7. Example video frames from an alternative complex outdoor environment with their corresponding pixel-by-pixel calls vs truth with and without applying the Siamese technique during training.

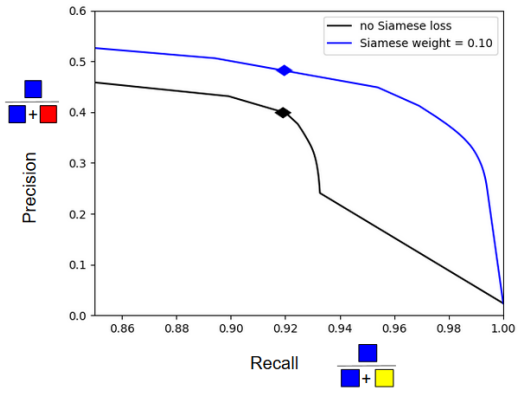

Fig. 8. Precision/Recall curves for the outdoor city environment. Showing only the final $w_s = 0.1$ curve and the curve without the Siamese technique for clarity.

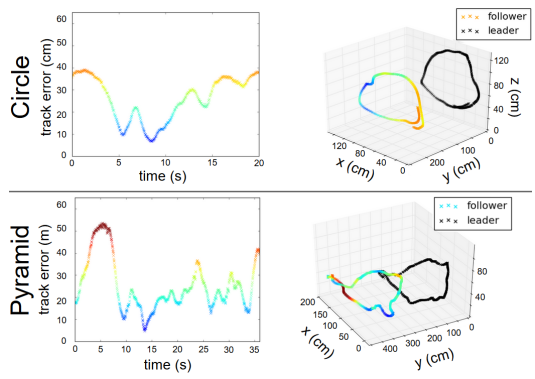

Fig. 9. Track error vs time and position of both quadcopters vs time

Our quadcopter detection, tracking, and velocity control program runs at 24.0 ± 3.4Hz on the onboard ODROID XU4's CPU. We chose a video resolution of 96 $\times$ 72 to achieve 24 Hz. We use the full field of view of the camera. This setup allows reliable formation control for quadcopters ¡ 4 meters apart. Beyond 4 meters, the quadcopter is not well resolved. Running at larger ranges would require finer resolution. Running with a GPU would make this possible in real time - this is left to a follow-on study.

VI. CONCLUSIONS

We have introduced a novel neural network training method that mitigates persistent false clutter detections common in CNN video processing. This method takes inspiration from the Siamese networks technique, penalizing the network's loss for generating dissimilar consecutive heatmaps.

We have implemented this "temporal Siamese clutter mitigation" in a fiducial-free vision-based quadcopter formation control guidance algorithm. Our successful leader-follower quadcopter formation control flight tests using this method show track errors of 30 cm: less than the size of the quadcopters themselves. Video processing was done in real time at 24Hz onboard the quadcopter's ODROID-XU4's CPU. This capability makes formation control possible in the presence of electronic jamming, indoors, or anywhere else that GPS and/or wireless communications are denied.

The following are potential follow up studies, each with their own inherent challenges. Performance in a more diverse set of environments (including outdoors) with different quadcopters could be investigated. Using the ODROID XU4's *GPU* to run the Siamese Hourglass CNN in real time, or simply implementing it on a different single-board computer could also boost the video resolution beyond 96×72 and improve tracking results. Additional cameras and/or a fisheye lens could be used to enlarge the quadcopters' field of regard. Formation control with more than 2 quadcopters could be attempted. Modifications to the CNN architecture, including processing multiple frames at-a-time, could be tested.

REFERENCES

[1] P. T. Nathan, H. A. F. Almurib, and T. N. Kumar, "A review of autonomous multi-agent quad-rotor control techniques and applications," in *2011 4th International Conference on Mechatronics (ICOM)*, 2011, pp. 1–7.

[2] Z. Hou, W. Wang, G. Zhang, and C. Han, "A survey on the formation control of multiple quadrotors," in *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2017, pp. 219–225.

[3] K. E. Wenzel, A. Masselli, and A. Zell, "Visual tracking and following of a quadrocopter by another quadrocopter," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, pp. 4993–4998. DOI: 10.1109/IROS.2012.6385635.

[4] V. Walter, N. Staub, A. Franchi, and M. Saska, "Uvdar system for visual relative localization with application to leader–follower formations of multirotor uavs," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2637–2644, Jul. 2019, ISSN: 2377-3774. DOI: 10.1109/LRA.2019.2901683.

[5] R. Tron, J. Thomas, G. Loianno, J. Polin, V. Kumar, and K. Daniilidis, "Vision-based formation control of aerial vehicles," English (US), in *RSS Robotics Science and Systems*, Jul. 2014.

[6] M. B. Vankadari, K. Das, and S. Kumar, "Autonomous leader-follower architecture of a.r. drones in gps constrained environments," in *Proceedings of the Advances in Robotics*, ser. AIR '17, New Delhi, India: Association for Computing Machinery, 2017, ISBN: 9781450352949. DOI: 10.1145/3132446.3134915. [Online]. Available: https://doi.org/10.1145/3132446.3134915.

[7] M. Saska, "Mav-swarms: Unmanned aerial vehicles stabilized along a given path using onboard relative localization," in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, Jun. 2015, pp. 894–903. DOI: 10.1109/ICUAS.2015.7152376.

[8] M. Saska, T. Báča, J. Thomas, J. Chudoba, L. Preucil, T. Krajník, J. Faigl, G. Loianno, and V. Kumar, "System for deployment of groups of unmanned micro aerial vehicles in gps-denied environments using onboard visual relative localization," *Autonomous Robots*, vol. 41, Apr. 2016. DOI: 10.1007/s10514-016-9567-z.

[9] T. Krajník, M. A. Nitsche, J. Faigl, P. Vanek, M. Saska, L. Preucil, T. Duckett, and M. Mejail, "A practical multirobot localization system," *Journal of Intelligent and Robotic Systems*, vol. 76, pp. 539–562, 2014.

[10] F. Schilling, J. Lecoeur, F. Schiano, and D. Floreano, "Learning vision-based flight in drone swarms by imitation," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, Aug. 2019. DOI: 10.1109/LRA.2019.2935377.

[11] M. Vrba and M. Saska, "Marker-less micro aerial vehicle detection and localization using convolutional neural networks," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2459–2466, 2020.

[12] K. Nakamura, J. Hao, S. Yamamoto, and T. Itoh, "Document image segmentation into text, continuous-tone and screened-halftone region by the neural networks.," in *IAPR Workshop on Machine Vision Applications*, 1996, pp. 450–453.

[13] I. Melekhov, J. Ylioinas, J. Kannala, and E. Rahtu, "Image-based localization using hourglass networks," in *The IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct. 2017.

[14] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4724–4732.

[15] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *European conference on computer vision*, Springer, 2016, pp. 483–499.

[16] E. Benson, M. P. Pound, A. P. French, A. S. Jackson, and T. P. Pridmore, "Deep hourglass for brain tumor segmentation," in *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, A. Crimi, S. Bakas, H. Kuijf, F. Keyvan, M. Reyes, and T. van Walsum, Eds., Cham: Springer International Publishing, 2019, pp. 419–428, ISBN: 978-3-030-11726-9.

[17] P. Li, Y. Lin, and E. Schultz-Fellenz, *Contextual hourglass network for semantic segmentation of high resolution aerial imagery*, 2018. arXiv: 1810.12813 [cs.CV].

[18] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–I.

[19] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," in *Advances in neural information processing systems*, 1994, pp. 737–744.

[20] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, vol. 2, 2015.

[21] S. Zagoruyko and N. Komodakis, "Learning to compare image patches via convolutional neural networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015.

[22] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. S. Torr, "Fully-convolutional siamese networks for object tracking," in *Computer Vision – ECCV 2016 Workshops*, G. Hua and H. Jégou, Eds., Cham: Springer International Publishing, 2016, pp. 850–865, ISBN: 978-3-319-48881-3.

[23] J. Mueller and A. Thyagarajan, "Siamese recurrent architectures for learning sentence similarity," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[24] L. Baraldi, C. Grana, and R. Cucchiara, "A deep siamese network for scene detection in broadcast videos," *arXiv preprint arXiv:1510.08893*, 2015.

[25] V. Vezhnevets, V. Sazonov, and A. Andreeva, "A survey on pixel-based skin color detection techniques," Mar. 2004.

[26] P. Kakumanu, S. Makrogiannis, and N. Bourbakis, "A survey of skin-color modeling and detection methods," *Pattern Recognition*, vol. 40, pp. 1106–1122, Mar. 2007. DOI: 10.1016/j.patcog.2006.06.010.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2014.