Received April 5, 2021, accepted April 25, 2021, date of publication May 13, 2021, date of current version June 15, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3080136

YOLBO: You Only Look Back Once–A Low Latency Object Tracker Based on YOLO and Optical Flow

DANIEL S. KAPUTA, (Member, IEEE), AND BRIAN P. LANDY
Rochester Institute of Technology, Rochester, NY 14623, USA

Corresponding author: Daniel S. Kaputa (dskiee@rit.edu)

This work was supported in part by the Moog Space and Defense Group.

ABSTRACT One common computer vision task is to track an object as it moves from frame to frame within a video sequence. There are a myriad of applications for such capability and the underlying technologies to achieve this tracking are very well understood. More recently, deep convolutional neural networks have been employed to not only track, but also to classify objects as they are tracked from frame to frame. These models can be used in a tracking paradigm known as tracking by detection and can achieve very high tracking accuracy. The major drawback to these deep neural networks is the large amount of mathematical operations that must be performed for each inference which negatively impacts the number of tracked frames per second. For edge applications residing on size, weight, and power limited platforms, such as unmanned aerial vehicles, high frame rate and low latency real time tracking can be an elusive target. To overcome the limited power and computational resources of an edge compute device, various optimizations have been performed to trade off tracking speed, accuracy, power, and latency. Previous works on motion based interpolation with neural networks either do not take into account the latency accrued from camera image capture to tracking result or they compensate for this latency but are bottlenecked by the motion interpolation operation instead. The algorithm presented in this work gains the performance speedup used in previous motion based neural network inference papers and also performs a novel look back operation that is less cumbersome than other competing motion interpolation methods.

INDEX TERMS CNN, classifier, detector, neural network, low latency, tracker, UAV, YOLO, look back, drone, image processing.

I. INTRODUCTION

Computer vision [CV] and deep learning [DL] are enabling technologies behind advanced driver-assistance systems [ADAS], augmented and virtual reality [AR/VR] devices, UAV collision avoidance systems, and a myriad of other applications not yet realized. This paper focuses on a subset of computer vision which pertains to the algorithms that are used for object detection, object tracking, and object classification within video sequences. These terms are very inter-related and it is worth taking the time to clearly define them. Object detection can be performed on a single image and it is akin to the "Where's Waldo" books where one must find Waldo somewhere on the page. Object tracking is different in that the goal is to determine the location of an object as the object moves from video frame to video frame. The most obvious way to perform object tracking would be to simply perform object detection on each frame and is referred to as tracking by detection [TBD]. This is an inefficient implementation however and a more efficient approach involves sparser detection combined with traditional tracking methods. Not only is TBD inefficient but how does one reconcile the scenario where a tracked unmanned aerial vehicle [UAV] flies behind a tree? The answer is that techniques other than TBD can be used such as determining an object's trajectory and then inferring the future position of the object even when it becomes occluded. The field of determining an object's motion trajectory is quite deep and is actually the basis for many kinds of video compression algorithms such as MPEG encoding. In this paper the concept of using motion information to track an object is sometimes referred to as motion interpolation or motion marching as motion vectors are strung together to march an object's location from frame to frame. Lastly, object classification refers to determining the class that an object in an image belongs to. For example, did a UAV fly into a restricted air space or was it just an unladen swallow.

The associate editor coordinating the review of this manuscript and approving it for publication was Zijian Zhang.

The three above mentioned computer vision tasks have traditionally relied on techniques such as matched filtering, motion vectors, edge detection, principal component analysis, and other well developed methods. Recently, with the advent of deep convolutional neural networks, the vision scientist can pick and choose how to combine traditional and artificial intelligence [AI] based algorithms in order to achieve the desired result. Traditional algorithms such as the matched filter can take less computation time than a convolutional neural network [CNN] however they do not typically possess the scale and rotational invariance that a well trained CNN does. Lin *et al.* [1] came up with a way to combine the benefits of traditional methods with those of the newer AI methods in order to find the sweet spot of object detection, tracking, and classification. The approach entails performing CNN inference on inference frames [I frames in this work] and object tracking via traditional methods on extrapolation frames [M frames in this work]. This technique requires one to determine a skip factor which ultimately sets the ratio between I and M frames and proportionally scales the frames per second [FPS] that can be achieved. This frame rate speedup is not without a cost however as there is a slight drop in tracking accuracy as motion error is accrued. Skipping to speedup tracking throughput is a concept seen in the work of Zhu *et al.* [2], Ujiie *et al.* [3], Meenakshi *et al.* [4], and Buckler *et al.* [5].

The main contribution of this work is YOLBO, a tunable low latency lightweight tracker based on a CNN object detector and augmented with optical flow motion vectors that is capable of running on an edge embedded system with no cloud offloading. Our algorithm makes use of a look back technique which effectively marches forward previous "stale" CNN inferences by means of stored optical-flow-generated motion vectors. This algorithm does not rely on hardware specific video encoding/decoding capabilities which allows it to be readily ported to Graphics Processing Unit [GPU] or Field Programmable Gate Array [FPGA] edge compute devices. The optical flow based ROI-to-centroid tracking technique of this article reduces motion interpolation processing cost as compared to similar works and smooths out the overall processing frame rate by pre-calculating motion adjustments for different objects previously identified by the tracker. YOLBO is profiled on a standard desktop PC as well as on an Nvidia Jetson TX2 with different CNN models to show the benefits compared to the non-motion baseline inference case.

An example that demonstrates the benefits of YOLBO would be a scenario where images are being processed at 30 FPS on-board a UAV, that is one new frame every 33 ms. It also takes a full frame update from photon capture to read out of the frame into the on-board processor where it can be read. Traditional algorithms will therefore take 2 frame updates [66 ms] before adjusting the flight control surfaces based on visual information. For a drone flying at 180 mph [6] which is roughly 80 m/s, a UAV will travel 4.84 meters before making an action based on what it sees. For sense and avoid scenarios when flying through cluttered environments, operating on information that is 30 feet old is not an ideal scenario. YOLBO would allow for a UAV to cut the tracking latency in half and to act based on information that is only 15 feet delayed.

II. RELATED WORK

There is a need for a low latency, high frame rate, embedded object tracker and classifier for visual serving of UAVs for many applications such as UAV formation flying, collision avoidance, target acquisition, and more. It does not make sense to run the object tracker and the object classifier at the same rate as objects typically move much faster than they change class. By leveraging a traditional object tracker that is updated via deep learning key frames, the best of both worlds is possible as the classifier can be run at a slower rate and its scale and shift invariance capability effectively reloads the traditional tracker on every key frame.

This traditional computer vision and deep learning fusion finds a balance between frame rate, latency, and accuracy that new edge applications demand. The goal of this work is to improve on existing CNN and motion based trackers by significantly reducing the latency of the tracker without adding significant accuracy degradation. Great works with interpolation based object detection at the edge are performed by Zhu *et al.* [2], and Ujiie *et al.* [3], but both accrue large bulk latency delays due to the CNN inference time. Murray [7] summarizes this challenge best when he says, "…there is a trade-off when designing a tracker as a more powerful tracker gives more accurate detections, which facilitates the tracking, but at the same time runs slower which makes tracking harder since more frames must be skipped". The concept of using motion-based interpolation to reduce latency has been put to use to adjust 3D point cloud data in [8] by Nicolai *et al.* and also in a similar manner by Liu *et al.* [9] and Chen *et al.* [10] to combat network latency for remote inference.

In [9], data is sent to the cloud and inference time is relatively small resulting in very few M frames. This means motion marching is only for a small frame interval, and instead of logging inter-frame motion and marching it, [9] only calculates the motion from the inference frame to the final frame in one go by using reference picture select [RPS] to set the source frame to the frame of the cached detections. H.264 decoded motion vectors are generated for non consecutive frames, rather than logging the inter-frame differences and marching at the end. This succeeds due to temporal coherence between close frames however for network-denied edge applications with longer inference times this technique may not perform as well. A solution for devices with long inference times would need to take into account frame motion between many consecutive M frames and perform motion vector marching to preserve temporal continuity.

The closest implementation of look back motion marching is demonstrated in Glimpse [10], however there were some self reported issues in tracking due to the nature of the tracked feature selection due to device constraints but Glimpse sought to alleviate these with their methods. For example, many edge points are tracked for objects in a frame and all motion information is cached in an ‘‘active cache’’ until ‘‘stale’’ detections are received from a remote server. There is unfortunately too much motion information stored in the active cache to update the stale boxes to the current frame if every frame is processed. To mitigate against this scenario, [10] selects only a fraction of these cached frames which ‘‘swiftly catch up to the current frame, but might degrade performance since [they] discard many frames; a bigger [percent of cache] ensures reliable tracking, but it takes more time and [they] would be left with stale results’’, [10]. Because of the number of points tracked with the Glimpse region of interest [ROI] technique, and the size of the active cache, frames have to be removed from the latency reducing march operation, which lowers accuracy sooner over time when compared to using all possible motion information. But this information, may sometimes be worth removing as no motion may have been recorded. This is one of the ways Glimpse worked to solve the problem of latency reduction. In this article, Glimpse-like and ROI-Every-Frame will refer to ROI optical flow every frame, which presents the same issues that Glimpse sought to solve with their paper’s techniques. We present an alternative in this work.

Cintas *et al.* propose vision-based UAV tracking by another UAV by use of YOLOv3 and kernalized correlation filters [KCF] [11]. They recognized the need for deep learning key frames to effectively generate the new correlation templates that can be used in-between successive math-heavy inference-based key frames. The problem with the KCF approach [12] for target tracking is that they are very robust to translation changes of a target template within and image but are not immune to rotational or scale changes. Opromolla [13] uses YOLOv2 as well as bounding box refinement in order to better hone in on the centroid of another UAV flying in front of the tracking UAV. Opromolla however only scans the middle portion of the full image for targets to track which is not ideal in scenarios where a UAV must sense and avoid objects entering the scene from the top or bottom of the field of view.

Many others have exploited the temporally coherent properties of video, as mentioned by Feichtenhofer *et al.* [14], to improve object detection and tracking. A relevant field of study has formed which combines video based motion tracking with CNNs. These trackers use many traditional tracking techniques such as optical flow, motion vectors and simple macroblocks in combination with object detectors like YOLO.

For a UAV tracker that is simply monitoring ground based objects, the low latency advantage of the YOLBO tracker may not be necessary as the UAV is not flying at high speeds and a delay of 30 ms will not be very impactful to operation as seen in the work of Shao *et al.* [15]. For a future scenario though where thousands of UAVs are dog-fighting with each other at high speeds it will be important to not only follow a UAV with very low tracking latency but also to determine if that UAV is a friend or foe. YOLBO allows the modern UAV to track the present target with very high accuracy and low latency and also acquire new objects as they enter the field of view. Ultimately the vision control loop of the UAV will make a decision of whether or not to stay locked on an existing target or to modify its trajectory to jump to tracking a higher priority target. The capability to track and potentially follow multiple targets will become more and more important with the advent of military drone swarms as was demonstrated by the Indian army with a swarm of 70 UAVs [16].

Other applications involve high speed flight through a cluttered environment especially when there is no map or GPS signal. The fast, lightweight, and autonomous [FLA] program was developed by DARPA to focus on achieving this capability. Being able to not only recognize edges as is commonly done with Sobel filtering, but to be able to classify the edge as a tree leaf rather than a tree branch would mean the UAV might not have to change its trajectory. CNN-based classification allows for this secondary layer of contextual awareness that enables UAVs to make the best decision based upon their camera inputs. As mentioned in the introduction, the main contribution of this work is YOLBO, which is a tunable low latency lightweight tracker based on a CNN object detector and is augmented with optical flow motion vectors that is capable of running on an edge embedded system with no cloud offloading.

III. EDGE COMPUTING

A. EDGE DEVICES

There is a need to perform inference and localization on the edge especially for many military applications such as counter UAS. Some of the available edge compute topologies include ASICs, GPUs, FPGAs, CPUs, DSPs, and microcontrollers. Although any of these compute paradigms could be used, a typical size, weight, and power [SWaP] trade is performed on the technical side as well as a cost, schedule, and risk trade on the financial side. Most of the UAV trackers employ GPU based systems due to their low cost, familiar coding framework [CUDA], and relatively low SWaP numbers. Other parameters namely latency and frame rate must be considered though which point to FPGA-based systems overtaking GPU systems for embedded object tracking [17]. YOLBO was initially run on a desktop PC with an NVIDIA 1070ti GPU and then ported to the TX2 due to the straightforward and similar coding framework, however the ultimate goal is to port YOLBO to the Fusion 2 stereo camera and then to the Fusion 1 UAV [18] for UAV to UAV tracking.

B. LATENCY, FRAME RATE, ACCURACY

When designing a tracker, frame rate is an important characteristic to keep in mind. Frame rate is the number of frames per second and is the throughput of a computer vision system. Latency on the other hand is the time required to process a frame from photon capture to algorithm output and can be very different than frame rate as shown in Fig 1. Frame rate has been shown to increase when detected object locations are

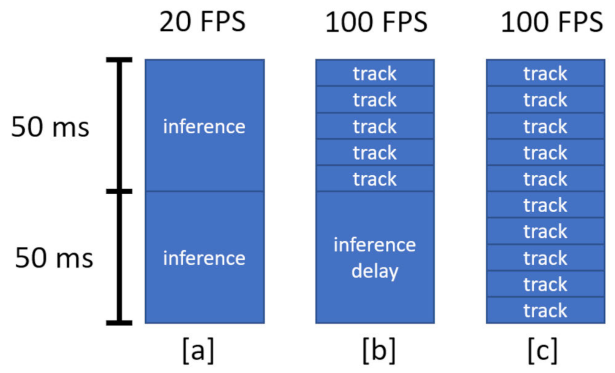


FIGURE 1. Latency without Motion Vectors [a], Latency with other Trackers [b], Motion Marching [YOLBO Tracker, Glimpse] [c].

interpolated with motion vectors as shown in [1] and [3] however these algorithms still suffer from a single inference bulk latency. YOLBO takes advantage of the frame rate speedup via motion-based interpolation however completely removes the bulk latency associated with competing algorithms.

IV. PROPOSED SYSTEM

The various components of the YOLBO tracker are presented below and are discussed as they pertain to the tracking application depicted in Fig 2. Our look back algorithm is similar to that described in [10] however ROI-to-centroid tracking as well as motion vector pre-calculation routines have been added to reduce computation overhead. An enhancement to the core YOLBO tracker is an IoU-based keep alive feature which is found in many other state of the art trackers. Lastly as mentioned above, YOLO and optical flow are used for object detection and motion interpolation.

A. OBJECT DETECTOR

The detectors used for this work come from the YOLO family of object detectors namely YOLOv2-Tiny, YOLOv3, and YOLOv4-Tiny. Although any object detection model could have been chosen, YOLO is a great start for edge inference. There are different reasons to chose YOLOv2-Tiny or YOLOv3, or even something like YOLOv4-Tiny and YOLOv4-Tiny_3L. Bulkier object detectors with more layers and learned filters may perform better with small objects. Using pruned or reduced versions may have less coefficients and infer faster, however, speed is not everything. If a system can not react to an object because it is not detected, then the speed at which the model performs detection is mute. YOLOv3 is a large model that takes a long time to run on the Jetson TX2 with native Darknet, but the look back based tracking and latency marching yield actionable results for small objects. YOLOv2-Tiny may be quick and require less skips, but is not as reliable for smaller objects. YOLOv3 was a good baseline for desktop operation as it was accurate and not too fast such that the benefits of skipping were not observable. YOLOv2-Tiny had decent accuracy performance on a PC and has the smallest weight size out of all, making it a good start for an FPGA based detector with limited on-chip memory. YOLOv4-Tiny yielded the highest FPS on the Jetson TX2 and the desktop environment and is considered state of the art in terms of light weight object detection.

B. MOTION ESTIMATION

Lucas-Kanade Optical flow motion estimation was the technique of choice for YOLBO due to its speed and ease of implementation with OpenCV. Various full frame, region of interest [ROI] and ROI-to-centroid tracking schemes were developed and are discussed in the methodology section below. Subdividing the image into discrete macroblocks was also attempted, however without a dedicated video decoder the required extra overhead was deemed too expensive. Optical flow block sizes of 8×8 pixels were used. For ROI generation of points, Harris Corner Detection is used.

C. CORE FEATURES

Look back tracking is fundamentally based around motion interpolation of object detections in a video sequence. Look back is aimed at reducing latency of tracking by adjusting outdated detection information with logged motion which is a similar idea to the operation of the work in [8], [9], and [10] and many more works. This motion logging and interpolation is a necessary part of look back, as the algorithm does not wait for CNN detections on frame N to finish before outputting a tracking result for frame N. A visual depiction of look back tracking can be seen in the frame sequence found in Fig 2. The top row of UAV images show the internals of the look back algorithm and the lower row of UAV images show the resultant bounding boxes derived from the look back algorithm. The colors are used for illustrative purposes only as the output bounding boxes for all frames are the same color in the actual implementation of YOLBO. Before diving into how the look back algorithm is different than the algorithms found in [3] and [10], some background nomenclature must first be introduced.

One common and well used method to speedup the frame rate of object detection via neural network inference is to perform inference on various key frames and to interpolate the motion of detected objects during the intermediate frames. In this work the inference frames are called I frames and marched or skip frames are called M frames. M frames are where existing information is interpolated from one frame to the next as in [2] and [3]. The variable N is the number of total frames that span a CNN inference time and it is mathematically equal to the number of M frames + 1 or [$N = M + 1$]. Fig 2 depicts a scenario where 2 frames are marched or skipped thus resulting in N with a value of 3.

The UAV enters the field of view in frame 0 and inference is started. This mode for the object is denoted as U which stands for untracked. Note that since motion interpolation is much faster than CNN inference, the inference does not finish until frame 3 [N]. While frame 0 is being processed by the neural network the object in the frame continues its motion which renders the inference result "stale", a term

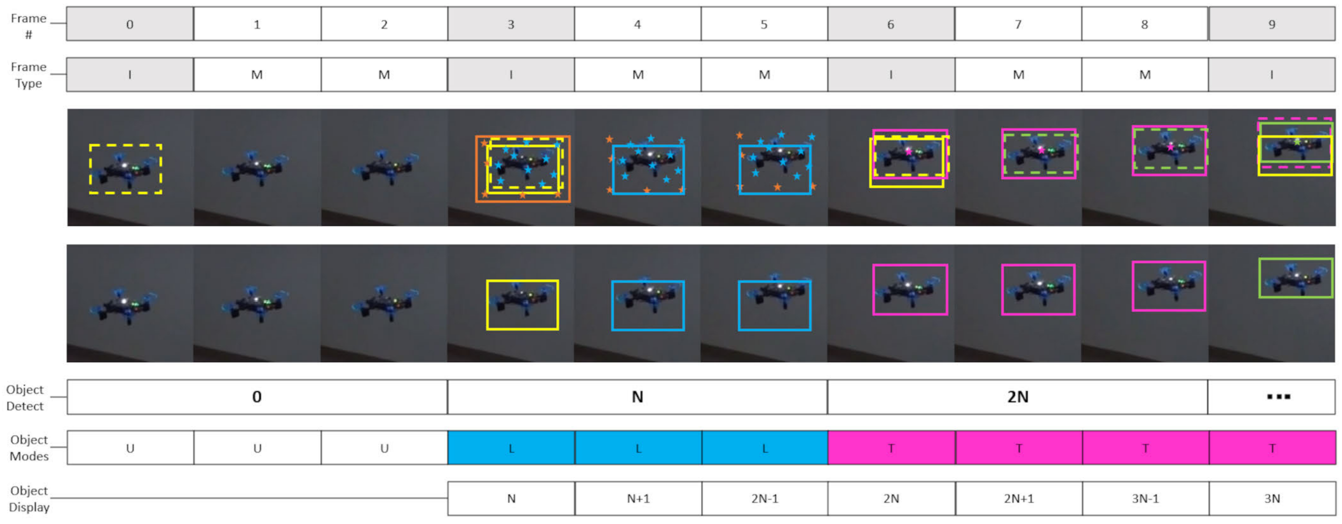


FIGURE 2. Look Back Video Sequence Details.

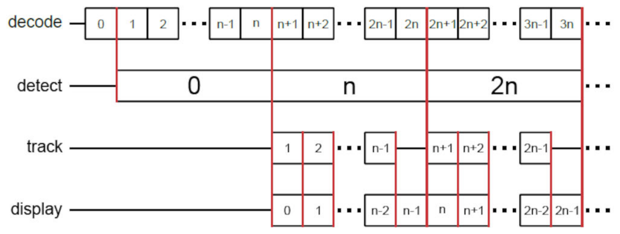


FIGURE 3. Method of Skipping Between Detections with Motion from [3].

coined in [10]. The solid yellow bounding box in frame 3 is at the same spatial location of the dashed yellow bounding box in frame 0 albeit stale or delayed by 3 frames. Many motion augmented AI-based trackers effectively use the solid yellow stale inference results and delay their frame display [by 3 frames in this case] so that the inference better lines up with the image of the object. This bulk delay of one full inference duration is conceptually depicted in Fig 1 and detailed in Fig 3 where frame 0 results do not appear until frame N. The bulk inference delay degrades tracking results as can be seen by comparing the location of solid yellow and dashed yellow bounding boxes within the same frame.

Once the frame 0 inference is complete and the solid yellow bounding box in frame 3 is produced, the object has transitioned into the locking or L mode. In this mode the look back algorithm understands that the location of the object is stale and that the actual object location should be within a certain tolerance of the stale bounding box. Look back grows the stale bounding box by roughly 20 percent and uses the enlarged bounding box [orange bounding box in frame 3] as a region of interest for the optical flow motion algorithm. Up to 15 key points are tracked by this optical flow implementation and they consist of the orange and blue stars in frames 3 [N], 4 [N+1], and 5 [2N−1]. Upon completion of the frame 3 inference as depicted by the frame 6 [2N] solid yellow bounding box, a search is performed to determine which of the 15 tracked points back in frame 3 originally lied within the dashed yellow bounding box of frame 3. These points which are shown as blue stars in frames 3, 4, and 5 are the points that are actually located on the object being tracked. This optical flow motion vector reduction technique makes sure that points not belonging to the object are not used for motion interpolation. Frames 3,4, and 5 are displayed as shown in the second row of UAV images however these bounding boxes are not very accurate due to the object not being fully tracked yet.

The next step after reducing the 15 tracked points down to a subset of marching points is to average all marching motion vectors [vectors associated with blue stars] down to a single motion vector and to march that forward to arrive at the solid purple "marched" bounding box. This solid purple bounding box in frame 6 is the location of an object determined via look back and it can be seen that it will be at a slightly different location than the dashed yellow inference bounding box in frame 6. The difference between these boxes manifests itself as the accuracy degradation based on motion vector errors of the look back approach compared to the CNN inference every frame approach.

Lastly, starting at frame 6, the object enters tracking or T mode where the purple bounding box centroid denoted with the purple star is marched forward every frame via single point optical flow. This transition from multiple point ROI based tracking as used in L mode into single point centroid tracking in T mode is a key feature [ROI-to-centroid tracking] that allows look back to run faster than ROI-every-frame approaches, and more like other mobile trackers such as [10]. At T mode, the bounding box is on the actual target as soon as the next frame is read in and the object can now be tracked with a centroid generated by its box position. With this centroid tracking data, the critical marching latency for a stale detection frame can be reduced and distributed over the past N number of frames as well. In frame 7 the motion vector from frame 6 [determined via optical flow on the purple centroid] to frame 7 [2N+1] is determined and the purple bounding box is displayed. In frame 8 [3N−1] however the motion vector is determined from frame 6 to frame 8. In this manner, by the time that frame 9 [3N] is reached a motion vector has been determined that links or marches from frame 6 to frame 9.

Instead of marching the purple bounding box from frame 6 to frame 9 [thus arriving at the dashed purple box in frame 9], there is new inference information available at frame 9 that is used to 'snap back' the motion tracking error. At frame 9, the yellow dashed bounding box from frame 6 is finished processing and the yellow dashed box is marched forward via the stored motion vector linking frames 6 and 9 to arrive at the green solid bounding box. This stored motion vector that skips multiple frames from 6 to 9 is the motion vector pre-calculation. The accumulated motion of the single tracked point used from 6 to 9 is used again and has already been calculated. This is better than the ROI method as it does not need to determine if several points are within a new bounding box thus resulting in greatly reduced marching latency. Note that the green dashed boxes in frames 7 and 8 are marched boxes and are not displayed or even calculated as the ROI-to-centroid vector addition from frame 6 to frame 9 skips over these frames. Fig 3 sourced from [3], shows the implementation of the tracker that does not have motion marching capability and the track of frame N is always off by one inference delay worth of time. With look back, this inference latency is cut out as the limiting factor.

D. KEEP ALIVE

The YOLBO tracker also uses a "keep alive" method that will keep an inferred object's box alive in memory for a certain number of frames after its association from one tracklet to the next fails. This feature mitigates faulty detections where an object is not picked up in the I frame but still exists in the video sequence. When an object has left the sequence possibly due to occlusion, small object size, poor network training, etc.., and the number of keep alive frames is exceeded, the object is subsequently set to F mode. YOLBO generates tracklets for N frames and each tracklet is linked to the next via IoU based matching. A unique ID is assigned to each matched object track for indexing purposes and unmatched objects receive a new ID. Objects that disappear are kept alive for a certain number of frames depending on the configuration settings. This keep alive feature pulls in the last detection result from a previous tracklet and re-uses it in case the CNN failed to predict a bounding box with enough confidence for an object in the next tracklet. This tracklet association process is explained in [3] and [7] and is detailed in [19] as well.

V. FRAMEWORKS AND DATASETS

A. DARKNET

For this implementation, Darknet, developed by Redmon [20] was chosen to be the object detection framework. The C language backbone of this framework is a good starting point for bare-metal work and it can be compiled and run without any GPU support. Along with the Darknet framework, several YOLO object detection models were put to use and the selected models [21]–[23] profiled are accessible to anyone and are sourced from the active forked version of Darknet [24]. The selected models were all trained on the same datasets and were left unmodified for the purpose of reproducibility. The MS-COCO and ImageNet trained models will be used to demonstrate the look back tracking algorithm. To this day, Darknet is an actively supported framework as well.

B. MS-COCO AND ImageNet

The models selected for this experiment were YOLOv2-Tiny, YOLOv3, YOLOv4, and YOLOv4-Tiny. All were trained using the MS-COCO dataset and started as ImageNet pre-trained weights. No special training for certain circumstances was performed for this work. The models used here are all sourced from, and can be found at [24]. The only modification to the models operation was a pruning of all non-people detections for the sequences.

C. OTB

To evaluate model performance, the Object Tracking Benchmark [OTB], by Wu *et al.* [25] was used. While YOLBO produces multi object tracking results, single object tracking sets were used for testing. The selected sequences from this set were any sequences which tracked a full figure of a person. This led to the selection of a large subset of OTB being used for testing. All results in the look back experimentation are compared to a best case scenario where motion tracking is not needed and an object detector is run on each frame. The metrics being reported on this set are bounding box overlap with the ground truth [IoU] and normalized centroid drift [NCD] from the ground truth to the predicted result. Since it is common for an IoU of 50% to be the threshold for a true positive [TP] detection, an IoU over 50% will serve as a consolidated summary of performance. This will allow for comparisons between models and skip counts to be made, but not a comparison to other OTB bench marked models. All models used for YOLBO are based on the unmodified MS-COCO and ImageNet models so they have already been bench-marked. For this analysis only the OTB sequences that contained tracked people were used.

VI. METHODOLOGY

A. DESIGN AND IMPLEMENTATION

The first steps to implementing the YOLBO algorithm was to choose appropriate frameworks to develop with in order to make baseline measurements. The plan was to implement look back with desktop GPUs and processors first and then move to a more optimized and low level implementation. Python was the language of choice as it is available for use on desktop PCs, the Nvidia Jetson TX2, and a variety of Xilinx Zynq UltraScale+ MPSoC platforms. This portability made python a good first choice as it was much more suitable for development as changes could be made quickly and with the aid of the python debugger. Development time has a trade off however as python has some performance limiting properties that could potentially disappear when switching to a more bare metal and deterministic application written directly in C or C++ with a reliable real time operating system.

The next step was to determine which object detection algorithm to use. Single shot detectors were chosen over region proposal based CNNs for speed reasons and performance with lower resources. Darknet is a prominent and well supported framework originally developed by [20] and maintained on a fork by [24]. After building all required libraries with and without GPU support, baseline inference was completed on a Desktop. Inference was done with python wrapped around a Darknet DLL with OpenCV used in various image processing, reading and display functions. Within OpenCV, optical flow was added to the inference loop to do inter-frame interpolation following the scheme implemented by [26]. Originally, this interpolation relied on full frame Lucas-Kanade optical flow. This was the basis for the frame skipping model and yielded a speedup as expected and shown in prior work. This approach was altered to cache motion information and perform look back to reduce latency per frame. Ground truth testing with OTB was added to the process to track accuracy, and timing information was logged to track and audit performance.

After all accuracy evaluations were complete, the application was modified for a multi-threaded environment where detections could be generated in the background and processing could still happen on the CPU. Python was a great rapid development tool, but the global interpreter lock proved to get in the way of true parallel processing. Multiprocessing replaced multi-threading as it allows 2 python process with 2 separate global interpreter locks to operate independently while still sharing relevant information in queues between each other. Multiprocessing is meant for compute bound applications and multi-threading is meant for I/O bound operations. Tracking frame rate continued to increase with these developmental changes and latency was reduced as well. Operation of the algorithm is based around python multiprocessing events.

B. NORMALIZED CENTROID DRIFT METRIC

When presenting results for a specific algorithm the first step is to clearly specify the metrics to be used. Most object detectors use an intersection over union metric to determine how accurate each inference is, and then deduce a mean average precision [mAP] value based on true positives as well as other factors such as false positives [FP] and false negatives [FN]. In this work, tracking accuracy is a comparison of the percent of true positive frames from the look back tracker to the base CNN inference which is run on every frame. For this reason the directive is to disregard the mAP metric and focus solely on the IoU TP metric and compare the degradation over skip amounts. Included as well in our analysis is the

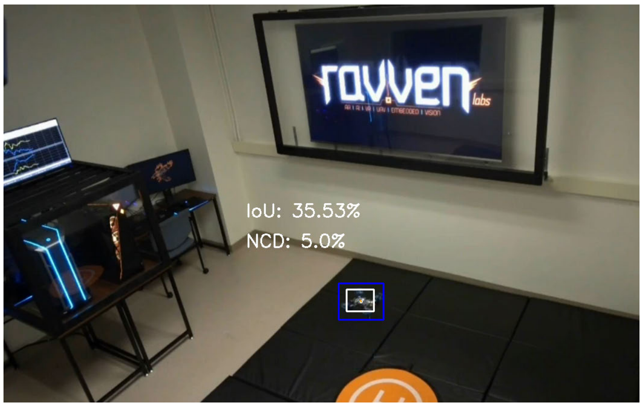


FIGURE 4. IoU vs NCD Errors.

normalized centroid drift metric. This is the distance vector magnitude from the predicted centroid to the ground truth bounding box centroid, with each component normalized by the ground truth bounding box width and height. The reason for using the NCD metric is that for UAV applications one cares about staying focused on the center of the object that the UAV is tracking. When using look back the predicted boxes are from previous frames, so as an object travels from the foreground to the background scale change can occur, however the centroid can still be accurate. The same applies to scale changes of an object due to shape; when tracking a walking person, outstretched arms can create a larger bounding box, but 10 frames later, the arms may be tucked in and the ground truth will be much narrower. The resulting IoU would most likely not result in a TP, but the centroids could be perfectly overlapped. In Fig 4 one can see the blue bounding box as the ground truth and the white bounding box is the inference. In this case the IoU is 35.53% however the tracked centroid is a match at exactly 5.0% with NCD. The inferred bounding box would be very accurate for closing visual tracking loops because the edges of the bounding box are not used, rather the center or mass or centroid of the object that is being tracked is used. For completeness sake, both IoU and NCD are logged. IoU true positive count and normalized centroid distance true positive count are averaged over all sequences and presented as a single value for consolidated result reporting. The aim is to show the degradation of TP frames over number of skips to illustrate the performance loss against latency benefit. This code base was written with development freedom and portability first in mind, and still does enough to show expected improvements due to the look back tracking algorithm, despite added overhead due to python.

C. YOLO EVERY FRAME BASELINE

To determine the baseline level of accuracy, latency, and frame rate, YOLO was run every frame on various OTB videos. This use case is what many pure inference applications typically run and it does not take into account any motion vectors or do any interpolation. It does benefit from

IoU based association between tracklets that can increase accuracy by linking previous detections forward to overcome detection flaws, but nothing is adjusted with motion. The accuracy and timing numbers for the YOLO every frame baseline can be seen by inspecting the Y axis values in Fig 5 corresponding to the X axis skip value of 0.

D. MOTION OFF

The motion off line in Fig 5 shows the worst case scenario where stale detections are not marched and there is no motion being recorded. This scenario is akin to performing inference on a decimated video sequence or simply throwing away frames between inference key frames. This motion off idea from [3] is great for visualizing the benefits and the maximum speed of operation for various interpolation techniques. The motion off line demonstrates the fastest possible processing time per frame and the worst case accuracy per frame as well. It will serve to demonstrate the ceiling and floor for accuracy and timing.

E. ROI OPTICAL FLOW EVERY FRAME

As seen in [10], every object in every frame has ROI based optical flow performed upon it. Fig 5 shows the timing and IoU results of the Glimpse-like ROI-every-frame method with varying skip amounts but no other Glimpse techniques like frame differencing. It is important to note that the ROI dimensions of the detections have a tunable margin for accepting motion information [10 pixels outward in all dimensions in this case] and that although accuracy could potentially be improved, the timing results would not change significantly. ROI based optical flow maintains accuracy while skipping, but takes more time in processing optical flow points while marching than the algorithm proposed for YOLBO. This is due to the ROI method needing relevant detection data in order to perform the motion march, and it processing more motion data than necessary. This delay before marching would be compounded on slower devices. The ROI-every-frame march is not as efficient and in the case of this experiment, performed worse than centroid based marching and tracking. The reasoning for this is that the ROI-Every-Frame approach uses more points to track deformable objects and this adds noise to the trajectory of tracked objects when all points are considered. This accuracy loss could potentially be mitigated by only accepting the largest group of motion points for tracking that move together with the same directional components.

F. FULL FRAME OPTICAL FLOW EVERY FRAME

Out of curiosity we explored full frame optical flow which turned out to be slower and less accurate than ROI based optical flow. As shown in Fig 5, when using full frame optical flow, the speedup due to frame skipping is severely bottlenecked by the march operation and is even worse than the Glimpse-like ROI-every-frame based tracker. This amount of processing produces negative returns after a certain point and is mentioned in detail in [10]. The same is true when using

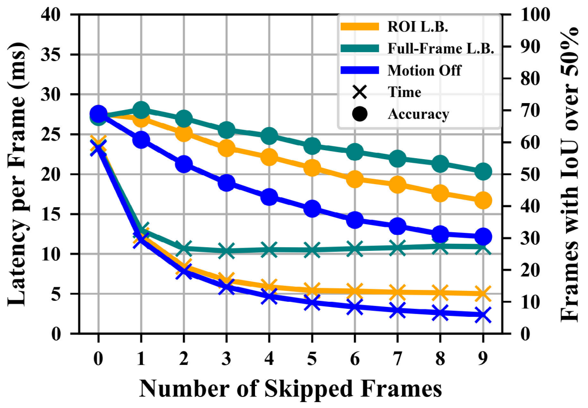


FIGURE 5. Recreated Results of ROI-every-frame Algorithm from [10] with YOLOv3 on Desktop PC.

ROI on each object in each frame, as in Fig 5, but to a lesser extent than full frame.

G. YOLO WITH LOOK BACK [YOLBO]

The next step was to run the exact same OTB videos through the YOLO look back algorithm with the desired frame skip. The optimal frame skipping rate was determined manually and was based on getting the most speedup for the minimal amount of accuracy loss. This look back method uses the ROI-to-centroid tracking improvement for reducing critical latencies as well. The pre-calculation benefit is demonstrated more when there are many objects in a frame being tracked. The results of the YOLBO tracker are discussed in the next section.

VII. RESULTS

YOLOv2-Tiny, v3, and v4-Tiny were all evaluated while using the look back tracking algorithm. Data was logged to compare the frame skip experiments to experiments with inference performed on each frame. Look back tracking YOLOv3 with a frame skip of 4 is compared to YOLOv3 on every frame in Fig 6 to observe the differences in NCD, frame read-to-output latency and IoU per frame. In this sequence comparison, a per-frame latency smoothing delay was introduced to lock the latency at 5 ms minimum per frame at frame 40. For a skip of 4 frames, per-frame latency was reduced 5 times as shown. Latency results, NCD, and IoU were tracked for all sequences and models. The average performance in terms of latency and accuracy for each frame skip of this look back based tracker for YOLOv3 on a PC is shown graphically in Fig 7. The YOLBO algorithm is shown in pink and the motion off results are shown in blue. The benefit of look back to frame latency is demonstrated and shown. For a skip of 1 frame, look back allows the inference delay to be distributed over 2 frames. In this scenario look back based tracking will never exceed a maximum latency, from frame read to output, of half the inference time. For a frame skip of 3, look back prediction distributes inference time over 3 frames and the maximum latency for a frame will not exceed one third of the inference time. This trend only continues

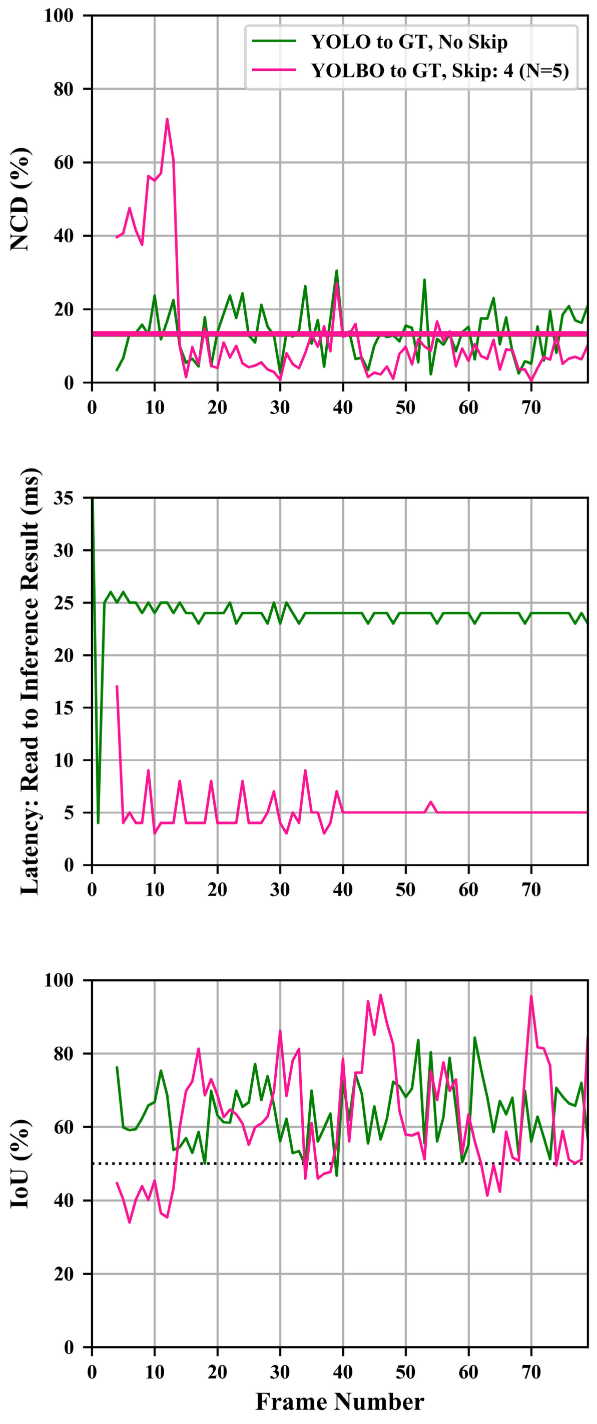


FIGURE 6. Metrics for Walking OTB Sequence with Basic Spin Lock Latency Smoothing Enabled at Frame 40.

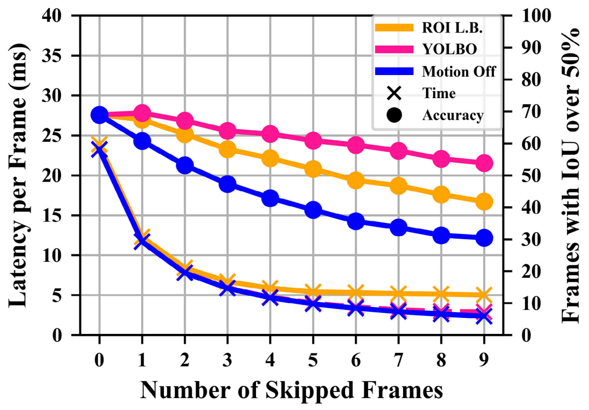


FIGURE 7. YOLOv3 Consolidated Performance on Desktop with Look Back.

TABLE 1. YOLOv2-Tiny with Look Back Results.

Skip	Latency (ms)	IoU TP (> 50%)	NCD TP (< 17.40%)	FPS (Hz)
0	20.30	48.62	48.62	49.26
1	10.56	50.21	52.03	94.69
2	7.10	48.69	50.20	140.80
3	5.68	46.76	48.92	176.06
4	4.36	46.60	48.37	229.36
5	3.74	44.05	46.80	267.38
6	3.30	43.51	47.60	303.03
7	2.99	41.99	47.50	334.45
8	2.84	41.62	47.45	352.11
9	2.73	40.14	45.76	366.30

as long as the other compute resources are able to keep up with the desired number of skip frames. Visualizations such as Fig 7 are important for determining how tracker skips should be decided. Accuracy fall off is mostly linear but model inference times have a decaying return that saturates. An ideal skip is one that achieves the performance benefit required without skipping more than needed. Some important test time characteristics of this model include a keep alive frame number of 10 frames, an expanded bounding box for ROI with a scale factor of 1.2, and IoU based association. The ROI bounding box expansion can be tuned for the speed of objects being tracked. Faster objects generally require a larger ROI zone as they can move further. In testing, 1.2x was a good size for acquiring objects, but this can always be adjusted. The frame display, object drawing and all other unnecessary operations have been turned off for this analysis.

The recorded results of YOLOv2-Tiny on this set of OTB test sequences is shown in Table 1. For NCD, the TP threshold was biased to a value that would generate the same number of TP frames as IoU with a skip of 0 (no skipping) for each model. IoU and NCD will have different degradation rates but started at the same number of TP samples. This illustrates the effect of bounding box scale variation on the IoU results, and allows NCD to add validity to higher frame skips because that accuracy metric is not affected by target to ground truth scale differences. The same evaluations were performed on a Desktop and averages were calculated for YOLOv3 as shown in table 2 as well as YOLOv4-Tiny in table 3. The results for YOLOv4 are shown in table 4 in order to illustrate the comparison between YOLOv4 and YOLOv4-Tiny. YOLOv4-Tiny, being the smaller model, is faster on a PC than YOLOv4 until about a skip of 5 is used. However, a combination of both YOLBO and model optimization is the goal for future usage.

TABLE 2. YOLOv3 with Look Back Results.

Skip	Latency (ms)	IoU TP (> 50%)	NCD TP (< 24.95%)	FPS (Hz)
0	23.24	68.88	68.87	43.48
1	11.70	69.46	71.93	85.4
2	7.81	67.09	69.24	128.04
3	5.90	63.88	66.97	169.49
4	4.73	62.91	66.15	211.42
5	4.01	60.83	63.41	249.38
6	3.50	59.48	62.52	285.71
7	3.17	57.66	60.84	315.46
8	2.97	55.14	59.43	336.70
9	2.91	53.84	59.57	343.64

TABLE 3. YOLOv4-Tiny with Look Back Results.

Skip	Latency (ms)	IoU TP (> 50%)	NCD TP (< 19.59%)	FPS (Hz)
0	4.82	64.64	64.64	207.47
1	3.13	66.30	68.70	319.50
2	2.87	62.28	65.19	348.43
3	2.78	60.41	62.87	359.71
4	2.69	58.59	61.42	371.74
5	2.66	55.65	58.79	375.94
6	2.61	52.57	57.72	383.14
7	2.58	51.16	56.87	387.60
8	2.70	49.46	56.30	370.37
9	2.60	48.52	54.46	384.62

TABLE 4. YOLOv4 with Look Back Results.

Skip	Latency (ms)	IoU TP (> 50%)	NCD TP (< 30.79%)	FPS (Hz)
0	27.56	68.06	68.08	36.29
1	13.81	69.00	69.37	72.39
2	9.53	65.97	67.46	104.98
3	6.93	64.01	67.10	144.20
4	5.52	62.30	65.19	181.15
5	4.66	60.43	63.53	214.81
6	4.02	58.42	62.27	249.02
7	3.71	55.87	61.44	269.69
8	3.59	53.36	59.24	278.61
9	3.45	53.73	60.55	289.74

VIII. LIMITATIONS

Optical flow processing and motion vector marching will eventually have diminishing accuracy returns as the skip size is drastically increased. Due to the nature of the latency minimization from look back this accuracy degradation will happen earlier than with normal optical flow interpolation between detection frames. This phenomena is because for each detection period the last frame before a re-detect has been marched 2*N-1 times rather than N-1 times as is done with standard motion interpolation schemes that do not minimize latency. The mitigation to this limitation is to be mindful of the skip number and to take into account the latency to accuracy ratio according to the skip number for one's desired application.

IX. CONCLUSION AND FUTURE WORK

This work demonstrates the usefulness of look back tracking for edge devices. The work builds upon many already successful tracking paradigms and is portable for different use cases and hardware topologies. Look back tracking with YOLOv3 in native Darknet, on a desktop PC, achieves a 4.91× latency reduction in exchange for a 5.97% IoU accuracy drop. When only looking at NCD from ground truth, this drop is only 2.72%. On the Nvidia Jetson TX2, YOLOv2-Tiny receives a 9.2× latency reduction and speedup in exchange for an 8.48% IoU accuracy drop and 2.86% increase in NCD error. Similarly, YOLOv4-Tiny on the TX2 received a 2.82× latency reduction for a 4.84% drop in IoU accuracy and 1.77% increase in NCD error. These benefits extend to power benefits as well, as look back tracking allows for sparser use of a high power usage compute resource on an edge device. Look back with the TX2 and GPU frequency reduction saved 7.35 W of power for an accuracy drop that corresponds to look back with a skip of 10 while still providing results at a consistent frame rate. The GPU frequency was able to be lowered without heavily impacting frame rate. This has large implications for edge device throughput, power usage and response time for UAV based lightweight trackers due to the less cumbersome centroid tracking.

A newer version of YOLBO is being created that integrates a dynamic frame rate feature as seen in [3] and [2]. This will allow for the frame rate to be tuned in relation to desired power draw, latency, accuracy, or a combination of several factors. This dynamic tuning capability is highly desirable for devices which have different power modes such as a UAV that is running low on battery and needs to get back to base but still needs to perform collision avoidance maneuvers. Additional work entails targeting YOLBO to an embedded FPGA SoC such as the Fusion 2 camera and performing quantization on the layers of YOLOv2 to achieve a further frame rate increase.

ACKNOWLEDGMENT

This article consists of general capabilities information that is not defined as controlled technical data under ITAR Part 120.10 or EAR Part 772.

REFERENCES

[1] L. Lin, B. Liu, and Y. Xiao, "An object tracking method based on CNN and optical flow," in *Proc. 13th Int. Conf. Natural Comput., Fuzzy Syst. Knowl. Discovery (ICNC-FSKD)*, Jul. 2017, pp. 24–31.

[2] Y. Zhu, A. Samajdar, M. Mattina, and P. Whatmough, "Euphrates: Algorithm-SoC co-design for low-power mobile continuous vision," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 547–560. [Online]. Available: https://ieeexplore.ieee.org/document/8416854/

[3] T. Ujiie, M. Hiromoto, and T. Sato, "Interpolation-based object detection using motion vectors for embedded real-time tracking systems," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2018, pp. 616–624. [Online]. Available: https://ieeexplore.ieee.org/document/8575254/

[4] G. N. Meenakshi Sundaram, "A survey on real time object detection and tracking algorithms," *Int. J. Appl. Eng. Res.*, vol. 10, pp. 8290–8297, Apr. 2015.

[5] M. Buckler, P. Bedoukian, S. Jayasuriya, and A. Sampson, "EVA^2: Exploiting temporal redundancy in live computer vision," 2018, *arXiv:1803.06312*. [Online]. Available: http://arxiv.org/abs/1803.06312

[6] K. Stephanson. (Jul. 2017). *The Drone Racing League Sets Quadcopter Speed Record.* [Online]. Available: https://www.guinnessworldrecords.com/news/commercial/2017/7/the-drone-r%acing-league-builds-the-worlds-fastest-racing-drone-482701

[7] S. Murray, "Real-time multiple object tracking—A study on the importance of speed," 2017, *arXiv:1709.03572*. [Online]. Available: http://arxiv.org/abs/1709.03572

[8] P. Nicolai, J. Raczkowsky, and H. Wörn, "Continuous pre-calculation of human tracking with time-delayed Ground-truth—A hybrid approach to minimizing tracking latency by combination of different 3D cameras," in *Proc. 12th Int. Conf. Informat. Control, Autom. Robot.*, 2015, pp. 121–130.

[9] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, Aug. 2019, pp. 1–16, doi: 10.1145/3300061.3300116.

[10] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proc. 13th ACM Conf. Embedded Netw. Sensor Syst.*, Nov. 2015, pp. 155–168, doi: 10.1145/2809695.2809711.

[11] E. Cintas, B. Ozyer, and E. Simsek, "Vision-based moving UAV tracking by another UAV on low-cost hardware and a new ground control station," *IEEE Access*, vol. 8, pp. 194601–194611, 2020.

[12] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 3, pp. 583–596, Mar. 2015.

[13] R. Opromolla, G. Inchingolo, and G. Fasano, "Airborne visual detection and tracking of cooperative UAVs exploiting deep learning," *Sensors*, vol. 19, no. 19, p. 4332, Oct. 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/19/4332

[14] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Detect to track and track to detect," 2017, *arXiv:1710.03958*. [Online]. Available: http://arxiv.org/abs/1710.03958

[15] Y. Shao, X. Tang, H. Chu, Y. Mei, Z. Chang, X. Zhang, H. Zhan, and Y. Rao, "Research on target tracking system of quadrotor UAV based on monocular vision," in *Proc. Chin. Autom. Congr. (CAC)*, Nov. 2019, pp. 4772–4775.

[16] D. Hambling. *Indian Army Shows off Drone Swarm of Mass Destruction*. Section: Aerospace & Defense. Accessed: Mar. 26, 2021. [Online]. Available: https://www.forbes.com/sites/davidhambling/2021/01/19/indian-army-shows%-off-drone-swarm-of-mass-destruction/

[17] M. Blott, T. Preusser, N. Fraser, G. Gambardella, K. O'Brien, and Y. Umuroglu, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," 2018, *arXiv:1809.04570*. [Online]. Available: http://arxiv.org/abs/1809.04570

[18] D. S. Kaputa and K. J. Owens, "Quadrotor drone system identification via model-based design and in-flight sine wave injections," in *Proc. AIAA Scitech Forum*, Jan. 2020, p. 1238, doi: 10.2514/6.2020-1238.

[19] E. Bochinski, T. Senst, and T. Sikora, "Extending IOU based multi-object tracking by visual information," in *Proc. 15th IEEE Int. Conf. Adv. Video Signal Based Surveill. (AVSS)*, Nov. 2018, pp. 1–6.

[20] J. Redmon. (2013). *DarkNet: Open Source Neural Networks in C*. [Online]. Available: https://pjreddie.com/darknet/

[21] A. Bochkovskiy, C.-Y. Wang, and H.-Y. Mark Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*. [Online]. Available: http://arxiv.org/abs/2004.10934

[22] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," 2016, *arXiv:1612.08242*. [Online]. Available: http://arxiv.org/abs/1612.08242

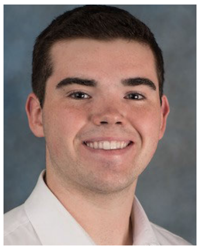
[23] Z. Wang, K. Xu, S. Wu, L. Liu, L. Liu, and D. Wang, "Sparse-YOLO: Hardware/software co-design of an FPGA accelerator for YOLOv2," *IEEE Access*, vol. 8, pp. 116569–116585, 2020.

[24] A. Bochkovskiy. (Jan. 2021). *AlexeyAB/Darknet*. Original-date: 2016-12-02T11:14:00Z. [Online]. Available: https://github.com/AlexeyAB/darknet

[25] Y. Wu, J. Lim, and M.-H. Yang, "Online object tracking: A benchmark," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2013, pp. 2411–2418.

[26] C.-E. Lin. (Apr. 2019). *Introduction to Motion Estimation with Optical Flow*. [Online]. Available: https://nanonets.com/blog/optical-flow/

DANIEL S. KAPUTA (Member, IEEE) was born in Troy, IL, USA, in 1980. He received the B.S. degree in computer engineering, and the M.S. and Ph.D. degrees in electrical engineering from the State University of New York at Buffalo, in 2002, 2004, and 2007, respectively. He worked at industry for ten years at various mil-aero companies, such as Lockheed Martin Gravity Systems and the Moog Space and Defense Group. He is currently working as an Assistant Professor with the Department of Computer Engineering, Rochester Institute of Technology. He is also the Director of Ravven Labs (www.ravvenlabs.com). His research interests include FPGA-based embedded vision, artificial intelligence, unmanned aerial vehicles, and augmented and virtual reality. He is a member of AIAA. He has served for the Mathworks Advisory Board for several years. He is heavily involved with the use and development of model-based design tools.



BRIAN P. LANDY was born in Ridgewood, New Jersey, USA, in 1997. He received the B.S. degree in computer engineering from the Rochester Institute of Technology, Rochester, NY, USA, in 2020, where he is currently pursuing the M.S. degree in computer engineering.

He has worked at several internship positions as an Embedded Software Engineer with Philips and a Verification Engineer with Crestron Electronics. He has participated in two research efforts with the RIT Machine Intelligence Laboratory, while studying at RIT. These include work with optical character recognition and Neural Sign Language Translation. In his final year and a half at RIT, from 2020 to 2021, he joined Ravven Labs to work on embedded computer vision applications and drone-focused systems.