Using YOLO and Recurrent Layers for Visual-Servoing in Underwater Environments

Peter Henderson Department of Computer Science
McGill University
Montreal, QC
peter.henderson@mail.mcgill.ca

Abstract

We present a robust multi-robot convoying approach relying on visual detection of the leading agent, thus enabling target following in unstructured 3-D environments. Our method is based on the idea of *tracking-by-detection*, which interleaves image-based position estimation via temporal filtering with efficient model-based object detection. The detection methods use a compressed version of the You Only Look Once convolutional bounding box architecture. We further expand on this with recurrent and causal convolutional temporal layers in attempt to improve results by taking into account both spatial and temporal features. We find that accuracy, recall, and precision are boosted versus the baseline, but intersection-over-union of the bounding boxes is not significantly affected. We discuss possible reasons for this and extensions on the work. To illustrate our solution, we collected extensive footage of an underwater swimming robot in ocean settings, and hand-annotated its location in each frame. We evaluate our algorithm on train/test data from this dataset and run live experiments with the robot using the visual servoing tracking-by-detection method.

1 Introduction

Vision-based tracking solutions have been applied to robot convoy tasks in a variety of contexts, including terrestrial driving [1, 2], on-rails maintenance vehicles [3], and unmanned aerial vehicles [4]. In the underwater domain, while prior efforts have focused on following divers and targets in constrained environments [5, 6], this work demonstrates robust and accurate tracking performance within challenging in-ocean robot convoy scenarios by visual detection of the leading agent, achieved through *tracking-by-detection* which combines target detection and temporally filtered image-based position estimation. Our solution is built upon several autonomous systems for enabling underwater tasks for a hexapod robot [7, 8, 9, 10, 11], as well as recent advances in real-time deep learning-based object detection frameworks [12, 13].

In the underwater realm, convoying tasks face great practical difficulties due to highly varied lighting conditions and external forces on the robot. While previous work in terrestrial and aerial systems used fiducial markers attached to targets to aid tracking, we choose a more general tracking-by-detection approach that is trained solely on the natural appearance of the object being tracked. While this strategy increases the complexity of the tracking task, it also offers the potential for greater robustness to changing pose variations of the target in which its attached markers may not be visible.

We use an annotated dataset of underwater imagery of the Aqua family of hexapod amphibious robots [7], from both on-board cameras of a trailing robot as well as diver-collected footage. This work focuses on exploring this dataset using a modified version of YOLO compressed to run on the robotic embedded system and stacking temporal layers in an attempt to boost the performance of the detector. We look at adding recurrent layers (Long-Term Short Term Memory networks) as one possible temporal extension. We also briefly explore using causal convolutions which have gained

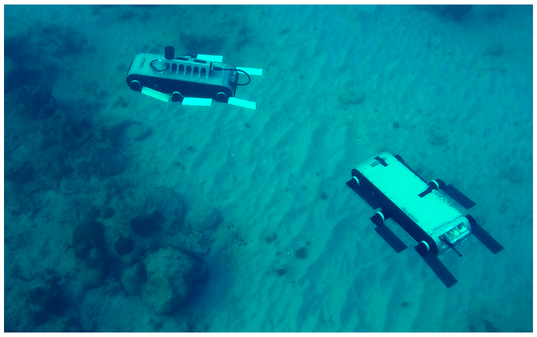
Figure 1: A sample still image from our underwater convoying field trial in Barbados using Aqua hexapods [7].

popularity after the success of the WaveNet architecture for audio generation [14]. We also use a modified version of YOLO [12], scaled down to run on-board the Aqua family of robots, in an open-water field trial to demonstrate the validity of our approach in real-world scenarios.

2 Related Work

2.1 Vision-Based Convoying

In convoying, there are several vision-based approaches which have shown to be promising in constrained settings. Some methods employ shared feature tracking to estimate all the agents' positions along the relative trajectory of the convoy, with map-sharing between the agents. Avanzini *et al.* demonstrate this with a SLAM-based approach [15]. These methods require communication between the agents which is difficult without specialized equipment in underwater agents. Other related works in vision-based convoying often employ template-based methods with fiducial markers placed on the leading agent [1, 2]. Such methods match the template to the image to calculate the estimated pose of the leading robot. While these methods could be used in our setting, we wish to avoid hand-crafted features or any external fiducial markers due to the possibility that these markers turn out of view of the tracking agent. Overall, existing methods require hand-crafted features or features that are highly invariant in the environment. We wish to avoid this. As such, we use convolutional neural networks and stacked temporal layers to try to learn from features which rely on the dataset, but aren't necessarily hand-crafted or constrained to one environment.

2.2 Detection Methods

Object detection is a task which has a myriad of approaches. Several benchmarks examine detection as a task in an effort to find the best algorithms. These include the PASCAL VOC challenges[16], the CalTech Pedestrians dataset[17], the COCO datasets [18], and many others. Many of the state of the art results on these benchmarks come from approaches which generally involve convolutional neural networks. While some approaches use region proposals to hone the convolutional neural networks (such as Fast-RCNN [19]), others use end-to-end convolutional neural networks in an effort to make a one-shot learning and detection system (the You Only Look Once (YOLO) [12] framework and the Single Shot Detector (SSD) [20] are two such example. Here, we focus on modified versions of the latter as they are more capable of running in real-time on a robotic system.

3 Background

3.1 YOLO

The YOLO [12] and YOLOv2 [13] architectures try to predict object bounding boxes in a one-shot end-to-end detection. This increases the speed of CNN based methods over other state-of-the-art methods such as Fast-RCNN [19] which require a separate layer for proposing regions of interest to train the detector on subpatches of the image. The methods generally work as follows (the differences between YOLO and YOLOv2 being mostly architectural). The method predicts and $S \times S$ grid of B bounding boxes. Each bounding box prediction generates a set of values consisting of: class prediction, confidence, (x, y) relative to the center of the grid cell, (width, height) relative to the image size. This method has shown to generate state-of-the-art results on the VOC datasets at real-time framerates. As such, we use this method as our base algorithm for the robot-detection system.

3.2 Temporal Network Layers

To take advantage of information across the temporal aspect of a video in an effort to boost detection accuracy, we make use of several temporal layers on top of the YOLO network. First, we examine using Long-Term Short-Term Memory networks [21] which can model sequence data through gated information updates. These have been used commonly in a wide range of problems with sequential data including language modeling [22] and even learning robotic sequences of actions for heart surgery [23]. Another popular method for modeling sequential data is causal convolutions. These are equivalent to convolutional neural networks with masks on inputs from sequences further than the unit's previous immediate connections. These can be augmented further with dilations of the kernel to take into account longer information contexts. These convolutional methods have been successfully used in the state-of-the-art for several tasks such as image generation [24], audio generation [14], and even machine translation [25]. As such, we attempt to make use of them to augment the YOLO outputs along the temporal aspect as well.

4 Implementation

4.1 DATASET COLLECTION

We collected a dataset of imagery of the Aqua family of hexapods during field trials at McGill University's Bellairs Research Institute in Barbados. This dataset includes approximately 5.2K third-person view images extracted from video footage of the robots swimming in various underwater environments, recorded from the diver's point of view and nearly 10K first-person point of view frames extracted from the tracking robot's camera front-facing camera.

We use the third-person video footage for training and validation, and the first-person video footage as a test set. We split the training data into two 80% and 20% sets for training and validation, respectively.

All images were hand-annotated with a ground-truth bounding boxes. We plan to publicly release this annotated dataset at `https://github.com/Breakend/AquaBoxDataset` for reproducibility and to enable future research in underwater robotics[1].

4.2 Modified YOLO Network

We used the YOLOv2 implementation [13], which has improved speed and accuracy performance over its predecessor formulation. Particularly, we configured the smaller and faster variant of the full model, `TinyYOLOv2`, as the detector needs to run onboard the robot at a reasonably fast frame rate. We therefore fine-tuned the 16 layers (depicted in Table 1) tiny model using our robot imagery dataset (described in our Dataset Collection section above). We use pretrained weights for the initial layers from the original YOLOv2 implementation.

Additionally, we condensed the `TinyYOLOv2` model for deployment on low-computing embedded robot platforms at reasonable frame rates, using the SqueezeNet strategy [26]. In particular, Ning *et al.* found through timing analysis that most of the overhead when running YOLO on CPU came from feeding forward through the convolutional layers[2]. Consequently, we:

- partially replace 3×3 filters with 1×1 filters, and
- decrease the number of input channels to 3×3 filters.

[1]Note the release is restricted for now due to IP rights and will be made available in the next year. Please email the author for access to the dataset if necessary.

[2]See Ning's online post titled "YOLO CPU Running Time Reduction: Basic Knowledge and Strategies" at `http://guanghan.info/blog/en/my-works/yolo-cpu-running-time-reduction-basic-knowledge-and-strategies`

Our reduced SqueezeNet-inspired YOLO architecture is described in Table 2. This architecture kept the same input resolution and approximately the same network depth, yet drastically decreased the number of filters for each layer. As we are using a network which has been designed for, and performs well on detection tasks of up to 9000 classes in the case of YOLO [13], the reduced capacity of the network should not significantly hurt the tracking performance for a single object class. Additionally, we used structures of two 1×1 filters followed by a single 3×3 filter, similar to the idea of Squeeze layers in Squeezenet [26], to compress the inputs to 3×3 filters. In addition, similarly to VGG [27] and the original YOLO architecture [12], we double the number of filters after every pooling step.

As in the original YOLOv2 configuration, both models employ batch normalization and leaky linear rectified unit activation functions [28] on all convolutional layers.

Type	Filters	Size/Stride	Output
Input			416×416
Convolutional	16	$3 \times 3/1$	416×416
Maxpool		$2 \times 2/2$	208×208
Convolutional	32	$3 \times 3/1$	208×208
Maxpool		$2 \times 2/2$	104×104
Convolutional	64	$3 \times 3/1$	104×104
Maxpool		$2 \times 2/2$	52×52
Convolutional	128	$3 \times 3/1$	52×52
Maxpool		$2 \times 2/2$	26×26
Convolutional	256	$3 \times 3/1$	26×26
Maxpool		$2 \times 2/2$	13×13
Convolutional	512	$3 \times 3/1$	13×13
Maxpool		$2 \times 2/1$	13×13
Convolutional	1024	$3 \times 3/1$	13×13
Convolutional	1024	$3 \times 3/1$	13×13
Convolutional	30	$1 \times 1/1$	13×13
Detection			

Table 1: Tiny YOLOv2 architecture

Type	Filters	Size/Stride	Output
Input			416×416
Convolutional	16	$7 \times 7/2$	208×208
Maxpool		$4 \times 4/4$	52×52
Convolutional	4	$1 \times 1/1$	52×52
Convolutional	4	$1 \times 1/1$	52×52
Convolutional	8	$3 \times 3/1$	52×52
Maxpool		$2 \times 2/2$	26×26
Convolutional	8	$1 \times 1/1$	26×26
Convolutional	8	$1 \times 1/1$	26×26
Convolutional	16	$3 \times 3/1$	26×26
Maxpool		$2 \times 2/2$	13×13
Convolutional	32	$3 \times 3/1$	13×13
Maxpool		$2 \times 2/2$	6×6
Convolutional	64	$3 \times 3/1$	6×6
Convolutional	30	$1 \times 1/1$	6×6
Detection			

Table 2: Reduced tiny YOLOv2 architecture

For training, we use the original YOLO training code provided by the authors and apply pre-processing augmentation (random bounded translations, rotations, and color channel changes) to the dataset to increase the variance in the data to prevent overfitting.

4.3 Recurrent Layers

In detection-based convoying, the system may lose sight of the object momentarily due to occlusion or lighting changes, and thus lose track of its leading agent. In an attempt to address this problem, we examine using recurrent layers stacked on top of the YOLO architecture, similarly to [29]. In their work, Ning et al. use the last layer of features output by the YOLO network for n frames (concatenated with the YOLO bounding box prediction which has the highest IOU with the ground truth) and feed them to single forward Long-Term Short-Term Memory Network (LSTM).

While Ning et al. assume that detection always takes place (as they test on the OTB-100 tracking dataset), we instead assume that the object may not be in frame in the dataset. Thus we make several architectural modifications to improve on their work and make it suitable for our purposes. First, Ning et al. use a simple mean squared error (MSE) loss between the output bounding box coordinates $(x_{center}, y_{center}, width, height)$ and the ground truth, with a penalty minimizing the MSE of the feature vector output by the recurrent layers and the feature vector output by YOLO. We find that in a scenario where there can be images with no bounding box predicted, this makes for an extremely unstable objective function. Therefore we instead use the objective function of YOLO modified for our single-bounding box single class case. This results in our final objective function being:

$$\begin{aligned}(I * \|\sqrt{w} - \sqrt{w_{pred}}\| + I * \|\sqrt{h} - \sqrt{h_{pred}}\|) * z_{coord} \\ + I * \|\hat{C} - C\| * z_{obj} \\ + (1 - I) * \|\hat{C} - C\| * z_{noobj} \quad (1)\end{aligned}$$

where $z_{coord}, z_{obj}, z_{noobj}$ are tunable hyper-parameters (left at 5, 1, .5 respectively based on the original YOLO objective), w, h, w_{pred}, h_{pred} are the width, height, predicted width and predicted height, respectively, $I \in \{0, 1\}$ indicates whether there exists the predicted object in the ground truth, C is the prediction of the confidence value and $\hat{C}$ is the Intersection Over Union (IOU) of the predicted bounding box with the ground truth.

Furthermore, to select which bounding box prediction of YOLO to use as input to our LSTM (in addition to features), we use the highest confidence bounding box rather than the one which overlaps the most with the ground truth. We find that the latter case is not a fair comparison or even possible for real-world use and thus eliminate this assumption.

In order to drive the final output to a normalized space (ranging from 0 to 1), we add fully connected layers with sigmoidal activation functions on top of the final LSTM output, similarly to YOLOv2 [13]. Redmon and Farhadi posit that this helps stabilize the training due to the normalization of the gradients at this layer. For these, we choose four fully connected layers of sizes: $\|yolo_{out}\|, 256, 32, 5$. We also apply dropout on the final dense layers at training time with a probability of .6 that the value is kept.

Lastly, we add multi-layer LSTMs to our experimental work as well as bidirectional LSTMs which have been shown to perform better on longer sequences of data [30]. A general diagram of our LSTM architecture can be seen in Figure 2.

Our recurrent detection implementation, based partially on code provided by [29], is made publicly accessible.[3] While they assume that the target of detection is always in the image, we do not make such an assumption and thus significantly modify their method, loss, and architecture. We use Tensorflow and Python for all the implementations recurrent implementations and the original C code for the YOLO training.[4]

4.4 Causal Convolutions

For our causal convolutional layers we use the same architecture as in WaveNet [14] as seen in Figure 3. We use only two residual blocks with a dilation rate of 1 and 2 respectively, kernel size 3 with latent dimensions being the size of the YOLO output vectors, except for the last layer which outputs the 5 predictions values (confidence, x, y, w, h). As in the architecture described in the paper,

[3]`https://github.com/Breakend/TemporalYolo`

[4]Can be found at: `https://pjreddie.com/darknet/yolo/`.

a residual block is formed from two convolutions combined through a $\tanh$ and σ non-linearity, with a residual connection to prevent gradient vanishing.

4.5 Training

For the recurrent methods, we take the pre-processed YOLO outputs from our trained YOLO network and train the temporal networks on this data with the original ground truth data as the labels. We train the network until our validation loss stops decreasing with an Adam optimizer and a learning rate of $1e^{-5}$.

4.6 VISUAL SERVOING CONTROLLER

The Aqua family of underwater robots allows 5 degree-of-freedom[5] control, which enables agile and fast motion in 3D. This characteristic makes vehicles of this family ideal for use in tracking applications that involve following other robots as well as divers [5].

One desired attribute of controllers in this type of setting is that the robot moves smoothly enough to avoid motion blur, which would degrade the quality of visual feedback. To this end we have opted for an image-based visual servoing controller that avoids explicitly estimating the 3D position of the target robot in the follower's camera coordinates, as this estimate typically suffers from high variance along the optical axis.

[5]Yaw, pitch, and roll rate, as well as forward and vertical speed

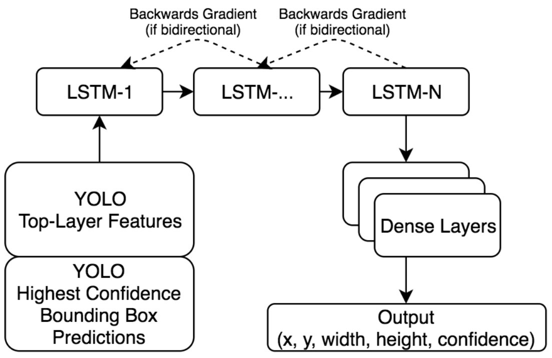
Backwards Gradient (if bidirectional)
Backwards Gradient (if bidirectional)
LSTM-1
LSTM-...
LSTM-N
YOLO Top-Layer Features
YOLO Highest Confidence Bounding Box Predictions
Dense Layers
Output (x, y, width, height, confidence)

Figure 2: Overview of the Recurrent-YOLO architecture. The original ROLO work [29] did not use bidirectional, dense layers, or multiple LSTM cells in their experiments.

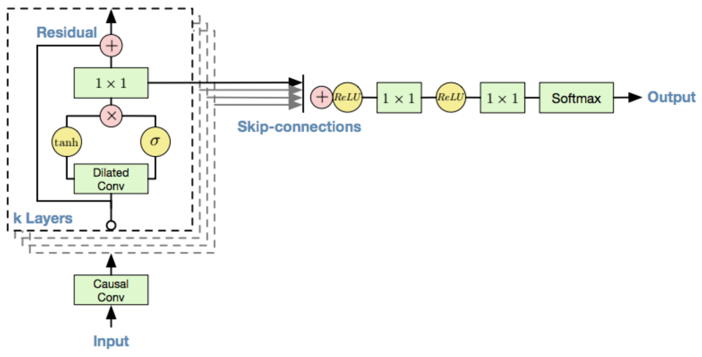

Figure 3: We use this diagram from [14] to illustrate the causal convolutional architecture we use. We only have two residual blocks with dilations 1 and 2 respectively.

This is of particular relevance in the underwater domain because performing camera calibration underwater is a time-consuming and error-prone operation. Our controller regulates the motion of the

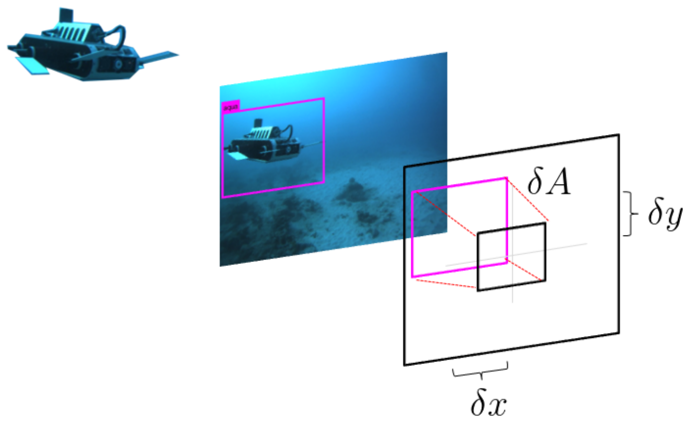

Figure 4: Errors used by the robot's feedback controller. δx is used for yaw control, δy for depth control, and the error in bounding box area, δA is used for forward speed control.

vehicle to bring the observed bounding boxes of the target robot on the center of the follower's image, and also to occupy a desired fraction of the total area of the image. It uses a set of three error sources, as shown in Fig. 4, namely the 2D translation error from the image center, and the difference between the desired and the observed bounding box area.

The desired roll rate and vertical speed are set to zero and are handled by the robot's 3D autopilot [31]. The translation error on the x-axis, δx, is converted to a yaw rate through a PID controller. Similarly, the translation error on the y-axis, δy, is scaled to a desired depth change in 3D space. When the area of the observed bounding box is bigger than desired, the robot's forward velocity is set to zero. We do not do a backup maneuver in this is case, even though the robot supports it, because rotating the legs 180^o is not an instantaneous motion. The difference in area of the observed versus the desired bounding box, namely δA, is scaled to a forward speed. Our controller sends commands at a rate of 10Hz, and assumes that a bounding box is detected at least every 2 seconds, otherwise it stops the robot.

5 Experiments and Results

We evaluate each of the implemented methods on the common test dataset described in the previous section using the metrics described below, with n_{images} the total number of test images, n_{TP} the number of true positives, n_{TN} the number of true negatives, n_{FN} the number of false negatives and n_{FP} the number of false positives:

- Accuracy : $\frac{n_{TP}+n_{TN}}{n_{images}}$
- Precision : $\frac{n_{TP}}{n_{TP}+n_{FP}}$
- Recall : $\frac{n_{TP}}{n_{TP}+n_{FN}}$
- Average Intersection Over Union (IOU) : Computed from the predicted and ground-truth bounding boxes over all true positive detections (between 0 and 1, with 1 being perfect)
- Localization failure rate (LFR): Number of true positive detections with IOU under 0.5 [32]
- Frames per second (FPS) : Number of images processed/second

Each of the implemented methods outputs its confidence that the target is visible in the image. This can be converted into a binary classification estimator by rounding off the confidence above 0.5. We note that this estimator is not formed by rounding off the IOU score, i.e. the classification output is not directly dependent on the accuracy of the bounding box localization.

We present the evaluation results in Table 3 for each of the algorithms that we considered. The *FPS* (frames per second) metric was measured across five runs on a CPU-only machine with a 2.7GHz

Algorithm	ACC	IOU	P	R	FPS	LFR
Tiny YOLO	**0.71**	**0.65**	**0.96**	**0.37**	0.91	**9.9%**
Reduced Tiny YOLO	0.63	0.54	**0.96**	0.20	**13.3**	39%
ROLO (n=3)	0.67	0.53	0.89	0.33	**278**	34%
ROLO (n=6)	0.68	0.53	0.89	0.35	135	31%
ROLO (n=9)	0.66	**0.54**	**0.96**	0.28	87	**28%**
ROLO (z=2,n=3)	**0.71**	0.51	0.88	**0.44**	205	37%
ROLO (z=2,n=6)	0.65	**0.54**	0.94	0.24	127	35%
ROLO (z=2,n=9)	0.67	**0.54**	**0.96**	0.29	78	36%
B-ROLO (z=1,n=3)	0.68	0.51	0.90	0.33	220	36%
B-ROLO (z=1,n=6)	0.66	0.51	0.93	0.27	121	42%
B-ROLO (z=1,n=9)	0.69	0.48	0.87	0.38	69	46%
B-ROLO (z=2,n=3)	0.67	0.49	0.88	0.32	195	38%
B-ROLO (z=2,n=6)	0.65	0.49	0.78	0.32	99	46%
B-ROLO (z=2,n=9)	0.66	0.47	0.90	0.32	64	49%
COLO (n=9)	0.66	0.49	0.87	0.27	89	35%

Table 3: Comparison of all tracking methods. Recurrent methods are labeled with ROLO. Bidirectional methods are indicated by B-ROLO. z indicates the number of LSTM layers, n indicates the numnber of frames in the sequence. COLO uses the output layers of YOLO but feeds them through the WaveNet causal convolutional architecture.

Intel i7 processor. For recurrent methods, *FPS* measures only the run time of the recurrent model using pre-computed Reduced Tiny YOLO features.

6 Discussion

6.1 Recurrent Methods

We trained our recurrent methods using the Reduced Tiny YOLO model's features precomputed on the previously described training set. We limit our analysis of ROLO to only use the reduced model's features as these can run closest to real time on our embedded robot system. Our results on the test set are shown in Table 3. For ROLO methods, z denotes the number of LSTM layers, n is the number of frames in a given sequence, and B-ROLO indicates bidirectional gradient updates. Note that the computational complexity presented here for ROLO methods is on top of the YOLO complexity and would most likely be bottlenecked by YOLO in all situations.

As can be seen in Table 3, recurrent layers on top of YOLO hone the confidence of the prediction with nearly no impact on the bounding box IOU. This honed confidence results in higher accuracy, precision, and recall. This can subsequently lead to greatly improved tracking as the robot is less likely to lose its leader. Furthermore, we believe that the error in both YOLO and ROLO is a result of noise in our training set. Our training set consisted of mostly GoPro collected footage from past years. The result are sequences of frames with the robot largely staying in the center of the frame. Our test set however was collected from on-board cameras of which footage was not available from past years. We believe that a larger training set with more on-board camera footage coupled with longer tracking sequences would benefit the ROLO/YOLO system. Furthermore, we find through inspection that many of the False Positive detections are of diver fins or hands. Upon closer review of the training set, we find that many of the bounding boxes contain these parts of the diver in them. We posit that the networks learn certain diver features to be part of the robot, albeit with a much lower probability,

Variations in layers and timesteps do not present a significant difference in performance with the exception that bidirectional methods typically converge to a much worse IOU with the ground truth. We would recommend that anyone use a recurrent layer to boost accuracy, precision and recall, but we do not expect that it will provide improvement in bounding box IOU with the provided objective as seen in our results. Furthermore, the frame rate impact is negligible with a single layer LSTM and short time frames, so we posit that it is only beneficial to use a recurrent layer on top of Reduced Tiny YOLO. While the best length of the frame sequence to examine may vary based on characteristics of the dataset, we suggest that $n = 3$ provides the best balance of speed, accuracy, IOU with ground truth, precision and recall. It boosts all of the metrics, while retaining IOU with the ground truth and keeping a relatively high FPS value. If some extra computational power is available, increasing layers of LSTMs can boost accuracy and recall further without impacting IOU or precision significantly. We attempted various re-balancing and re-weighting of the objective in our experiments and found that the presented settings worked best. We posit that an increase in IOU was not seen as the last feature layer of YOLO may not provide enough information to generate a more accurate bounding box. Future work to improve the recurrent system would target end-to-end experiments on both the convolutional and recurrent layers and experiments with different objective functions to boost the IOU as well as accuracy.

6.2 Causal Convolutional Methods

We see similar results as in the recurrent methods with causal convolutional layers. Part of the reason for this is that we only like at $n = 9$ frames due to time constraints. Causal convolutions with dilations are able to take much more context into account than this and the architecture we use is generally larger than the amount of data that is available and the amount of features available from the YOLO output layer. As such we don't see hugely beneficial results, though they are comparable to LSTM layers. We posit that with more data, end-to-end training and rearchitecting of the causal convolutional layers (along with taking into account $n > 9$ frames), we suspect causal convolutions will start to show more promise than LSTMs at $n \geq 9$ frames.

6.3 Field Trials

We also validated that the system works in practice on a real robot during field trials. We found that two robots were able to successfully convoy with the lead robot following a random path. A more detailed description of our field trial results can be found in [33].

7 Conclusion

Overall, we show that a real-time detector can be created using a modified YOLO architecture for the task of detecting an underwater robot. We further demonstrate that by training temporal layers on top of the YOLO outputs (taking into account a sequence of frames in a video) we can boost accuracy within acceptable framerate penalties. We validate that the system works in field trials on the robot and successfully perform 2-robot convoying using the *tracking-by-detection* visual servoing system. In the future, we want to train the system end-to-end to try to boost IOU performance from the temporal layer as well as the accuracy. We further want to apply the method to the CalTech pedestrians dataset with longer sequences of frames and a modified causal convolutional layer to take advantage of long contexts with more training data.

Acknowledgments

Some sections in this paper were borrowed from a conference submission by the author [33], with the permission of the instructor. This author did all the work on YOLO detection, recurrent layers, and causal convolutional layers (including implementation experiments and writing) and assisted with all other aspects of that work in the paper. This is a small part of a much larger work which evaluates several different trackers (including different convolutional architectures and frequency-based trackers) and conducts analysis on the field trials run with this system.

References

[1] Henry Schneiderman, Marilyn Nashman, Albert J Wavering, and Ronald Lumia. Vision-based robotic convoy driving. *Machine Vision and Applications*, (6):359–364, 1995.

[2] Carsten Fries and Hans-Joachim Wuensche. Monocular template-based vehicle tracking for autonomous convoy driving. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2727–2732. IEEE, 2014.

[3] Frederic Maire. Vision based anti-collision system for rail track maintenance vehicles. In *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*, pages 170–175. IEEE, 2007.

[4] Jacobo Jimenez Lugo, Andreas Masselli, and Andreas Zell. Following a quadrotor with another quadrotor using onboard vision. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 26–31. IEEE, 2013.

[5] Junaed Sattar and Gregory Dudek. Underwater human-robot interaction via biological motion identification. In *Robotics: Science and Systems*, 2009.

[6] Yonghui Hu, Wei Zhao, Long Wang, and Yingmin Jia. Underwater target following with a vision-based autonomous robotic fish. In *American Control Conference, 2009. ACC'09.*, pages 5265–5270. IEEE, 2009.

[7] Junaed Sattar, Gregory Dudek, Olivia Chiu, Ioannis Rekleitis, Philippe Giguere, Alec Mills, Nicolas Plamondon, Chris Prahacs, Yogesh Girdhar, Meyer Nahon, et al. Enabling autonomous capabilities in underwater robotics. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3628–3634. IEEE, 2008.

[8] Junaed Sattar and Gregory Dudek. Robust servo-control for underwater robots using banks of visual filters. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3583–3588. IEEE, 2009.

[9] Junaed Sattar and Gregory Dudek. A boosting approach to visual servo-control of an underwater robot. In *Experimental Robotics*, pages 417–428. Springer Berlin Heidelberg, 2009.

[10] Yogesh Girdhar and Gregory Dudek. Exploring underwater environments with curiosity. In *Computer and Robot Vision (CRV), 2014 Canadian Conference on*, pages 104–110. IEEE, 2014.

[11] David Meger, Juan Camilo Gamboa Higuera, Anqi Xu, Philippe Giguere, and Gregory Dudek. Learning legged swimming gaits from experience. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2332–2338. IEEE, 2015.

[12] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[13] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[14] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

[15] P. Avanzini, E. Royer, B. Thuilot, and J. P. Derutin. Using monocular visual SLAM to manually convoy a fleet of automatic urban vehicles. In *IEEE International Conference on Robotics and Automation*, pages 3219–3224, May 2013.

[16] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, (2):303–338, 2010.

[17] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: A benchmark. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 304–311. IEEE, 2009.

[18] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[19] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

[20] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[22] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Interspeech*, pages 194–197, 2012.

[23] Hermann Mayer, Faustino Gomez, Daan Wierstra, Istvan Nagy, Alois Knoll, and Jürgen Schmidhuber. A system for robotic heart surgery that learns to tie knots using recurrent neural networks. *Advanced Robotics*, 22(13-14):1521–1537, 2008.

[24] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

[25] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

[26] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, 2016.

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, 2014.

[28] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[29] Guanghan Ning, Zhi Zhang, Chen Huang, Zhihai He, Xiaobo Ren, and Haohong Wang. Spatially supervised recurrent convolutional neural networks for visual object tracking. *arXiv:1607.05781*, 2016.

[30] Alex Graves. Supervised sequence labelling. In *Supervised Sequence Labelling with Recurrent Neural Networks*, pages 5–13. Springer, 2012.

[31] David Meger, Florian Shkurti, David Cortés Poza, Philippe Giguère, and Gregory Dudek. 3d trajectory synthesis and control for a legged swimming robot. In *Proceedings of the IEEE International Conference on Robotics and Intelligent Systems (IROS)*, 2014.

[32] Luka Cehovin, Ales Leonardis, and Matej Kristan. Visual object tracking performance measures revisited. *CoRR*, 2015.

[33] Florian Shkurti, Wei-Di Chang, Peter Henderson, Md Jahidul Islam, Juan Camilo Gamboa Higuera, Jimmy Li, Travis Manderson, Anqi Xu, Gregory Dudek, and Junaed Sattar. Underwater multi-robot convoying using visual tracking by detection. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS) (in review)*, 2017.