“Improving fish tracking in FishVR”

Abstract:

In this document, we present a software and hardware pipeline for tracking 3D trajectory of multiple zebrafish in a fish tank. Then, we compare our results to the method that was previously used by our client. Finally, we discuss future steps for further improving the system.

1. Introduction

This project is a component of a larger project, Fish VR, which is concerned with creating a platform to determine and measure the relationship between visual stimuli and fish movement. An example could be projecting an image of a predator on the wall behind a fish tank, and studying the movements of the fish in reaction to it. Comprehending fish movement patterns has been used in applications in determining fish health, emotional states, stress levels, changing environment, and more [4, 5, 6, 7]. Obviously, to create such a system, the trajectory of the fish within the tank needs to be reliably measured, which is the focus of the project presented in this document.

Within a tank, fish move in a 3-dimensional space with fast and unpredictable movements; that necessitates a tracking technique to accommodate those properties. Traditional approaches of animal tracking have either been physically recorded by hand which leads to errors, or by attaching device(s) to an animal which can cause added stress and also has limitations based on the battery life of the device [9]. Additional tracking methods include tagging, sonar, 1 camera 2D tracking, 2 camera 3D tracking, and infrared [4, 5, 8]. For this research we have chosen to use a 2 camera 3D tracking implementation due to its low cost, good accuracy, and relatively low complexity.

2. Background & Related works

Methods for tracking fish are plentiful and each method has its appeals and disadvantages. Cheng et al prove that tracking in a 3D manner using two cameras can be achieved. Their method for tracking is by placing a camera in the vertical direction and the horizontal direction of the fish tank. Each camera is able to capture a 2D representation of where the fish is within the tank. These two cameras share a measurement on the X axis, combining these measures and

removing the redundant X measurement 3D movement can be tracked [5]. Pautsinaa et al., use a single infrared camera for 3D tracking pointed at a view from the top to the bottom of the tank. When processing the video output, fish position(s) in the X, and Y position can be processed based on the 2D view. To obtain the Z position the brightness (amount of reflection of infrared light) is compared against a known value and estimated [4]. The authors of this paper claim that this method is simple, effective in nearly all lighting conditions, however it lacks accuracy in tracking the Z position of the fish as the position is only an estimate.
To read more about the current state of research in zebrafish tracking, we recommend [m1] which provides a concise review of the literature. The conclusion in that paper is that tracking multiple zebrafish consistently remains a difficult and open problem in computer vision.

4. Data

To collect data, we recorded concurrent videos from two point-of-views of a fish tank: one horizontal view, and one vertical view. These videos were recorded at 1080p by cameras that were pointed approximately normal to the top and side planes of the fish tank, therefore allowing us to simply infer the cartesian coordinates (X,Y for the top view, and Z for the front view) and later combine those to create the 3-D coordinates.

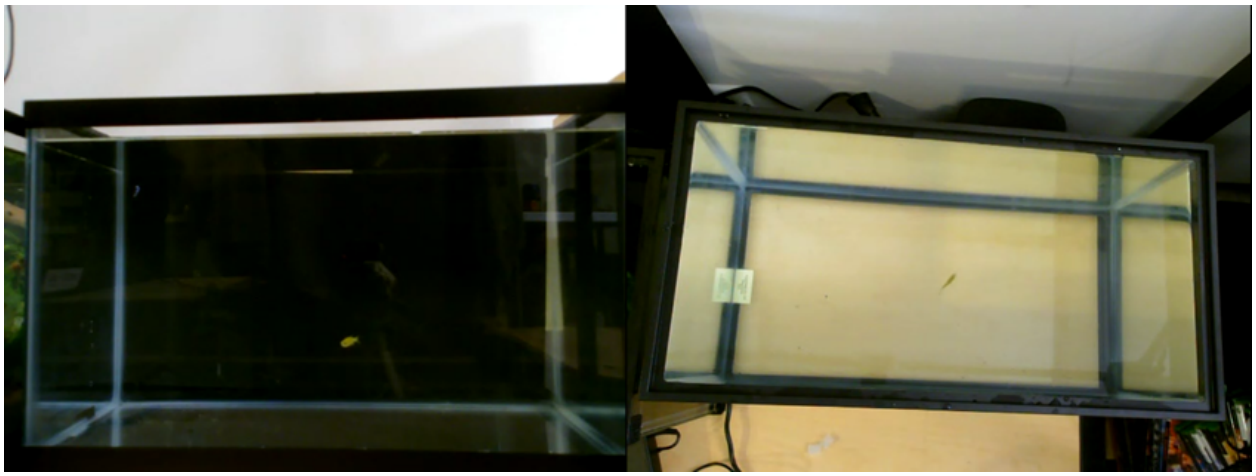

Figure 1. Left: snapshot of a horizontal (front view) video. Right: snapshot of a vertical (top view) video.

Due to the large file sizes, and constraints of available space on Google Drive, we were forced to downscale our videos to 480x640 resolution. It is possible that the low resolution and the blurrier image have had a negative impact on detection performance. Note that, in the finished project, data collection will be occurring in real-time, with the video stream used as input of the

detection and tracking algorithms (to be discussed later) for 3-D coordinate construction. Currently, however, we solely rely on pre-recorded videos of the fish for testing purposes.

5. Methods

5.1 Detection

We used YOLOv3 [m2] to perform object detection. As the name suggests, YOLOv3 is a more sophisticated and recent version of YOLO [m3]. The original YOLO divides each image into a S*S grid, and each grid cell is tasked with identifying B objects (B bounding boxes in each cell). For each detection, the following five parameters are detected: (1) the probability that the bounding box contains an object, (2) the X and (3) the Y coordinate of the center of the bounding box, (4) its width and (5) its height, and additionally, the probability that it belongs to each of the C classes that the network is trained on. Therefore, the output of the network is a tensor of dimension S*S*(5B+C) or a 5B+C vector for each grid cell. In the original version of YOLO, the input image is divided into a 7*7 (S=7) grid, and there can be at maximum two bounding boxes (detections) per grid cell (B=2). The number of classes, or C, depends on the dataset used for training (for COCO dataset, C = 80).

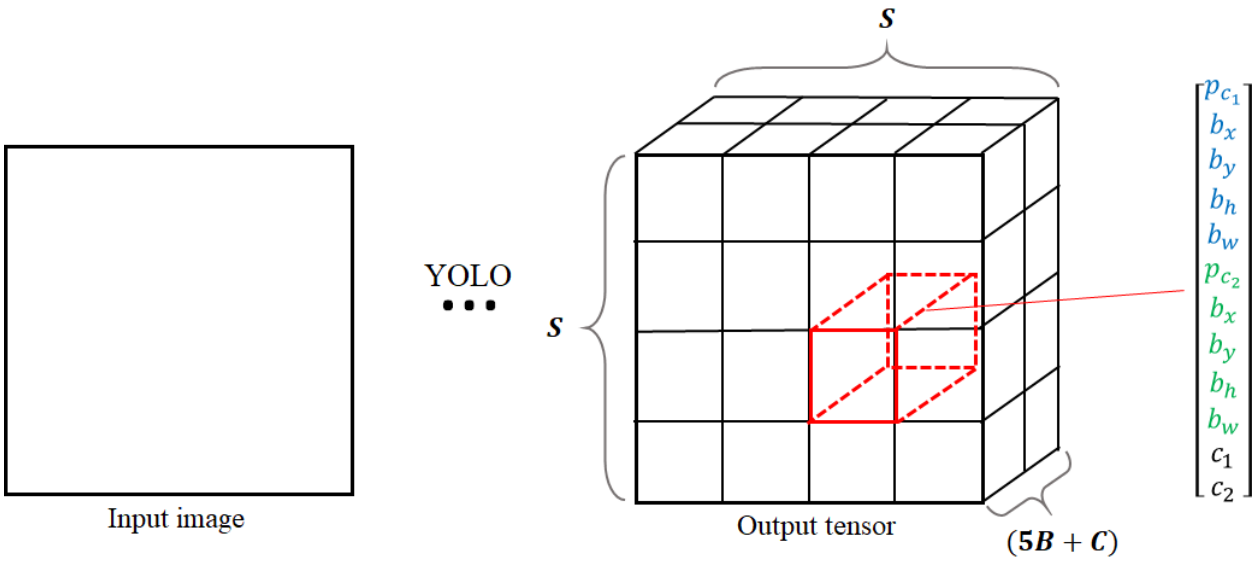

Figure 2. Schematic representation of input and output of YOLO

In contrast to older methods such as R-CNN [m4] which chose regions of interest (ROI) in the image and performed detection on each, the procedure used in YOLO effectively covers the entire image. The detection is done end-to-end using a single network with 24 convolutional layers (and two fully connected layers in the end) and each grid cell is processed in parallel, allowing for fast processing. As mentioned previously, YOLOv3 is more complex; it divides each

image into a 13*13 grid, uses three bounding boxes for each (B=3), and performs detection at three different scales (to pick up objects of different sizes) by downsampling the image in the network. In total, YOLOv3 has 106 convolutional layers.

5.2 Tracking

The problem of object tracking is defined as assigning a unique ID to each detected object in consecutive frames, where the tracking method uses the output of the detection algorithm to assign those IDs. The essence of successful tracking depends on tackling the so-called association problem: Given a set measurement (i.e. object detections in a frame), which ID should be assigned to which object.

For object tracking, two state-of-the-art algorithms are SORT [m5] and DeepSORT [m6]. SORT uses a linear Kalman filter with a constant velocity model for each bounding box to predict its state (location and dimensions) in the next frame. When detection is performed in the next frame, the pairwise intersection-over-union (IOU) of the predicted bounding boxes are compared to the bounding boxes in detection results, and a cost matrix is formed. The Hungarian algorithm is then used to assign an object ID to each bounding box. Intuitively, if the intersection of the two bounding boxes in consecutive frames is large, according to SORT, they should correspond to the same object. Understandably, if there is a gap in object detection for multiple frames, the IOU value when the object is detected again is very low, leading SORT to decide that the detection must correspond to a new object. Therefore, the performance of tracking depends to a large extent on detection performance.

DeepSORT utilizes the same idea but adds a neural network to generate features for each detected object and uses those in addition to the predictions by the Kalman filter. However there are two caveats,: (1) DeepSORT is pretrained on MARS dataset, and using a custom dataset is not as streamlined as the extremely popular networks such as YOLO. (2) Compared to human pedestrians, within the same species of fish, there are no distinct visual features that allow for distinguishing between them; this is exacerbated by the low resolution of our video capture.

Figure 3. Left: sample image of the same pedestrian in MARS dataset. Right: Zebrafish as recorded by the camera used in our setup.

In short, for our application, the neural network used in DeepSORT would be practically useless and we chose SORT instead.

6. Experimental Design

6.1 YOLOv3 implementation

For the YOLOv3 implementation, we used {https://github.com/ultralytics/yolov3} because it uses PyTorch and because it is currently the most up-to-date and popular YOLOv3 repository. The repository comes with pre-trained weights that are generated by training the YOLOv3 network on the COCO dataset [m7] (Common Objects in Context) which contains ~200,000 labeled images, divided between 80 classification classes. However, fish is not included in the dataset at all and even if it were, the sample images would likely not match what the cameras used in our setup capture. We confirmed this by running the pre-trained network on a sample video of the front view; as expected, the detection performance was extremely poor, and even when the fish was “detected”, it was labeled as a banana. To actually detect any of the zebrafish in our network, we had to train our network with our own labeled images.

Figure 4. Left: sample image of a banana in COCO dataset [m7]. Right: Fish detected as a banana using the pretrained network.

6.2 Labeling

By labeling, we mean having a rectangular bounding box around each fish in each frame of training video. We use CVAT (Computer Vision Annotation Tool), which unlike other free tools such as labelImg, accepts video files. Further, it provides basic object tracking functionality for simplifying the labeling process, though some cleanup is always needed.

We used the web version of CVAT to label and export a 90 second (2700 frames) portion of the captured video. Initially, we began by stepping frame-by-frame to label our zebrafish. We eventually discovered the built-in tracking tool and ran the algorithm for nearly an hour, which automatically generated the annotations around the fish in the majority of the frames.

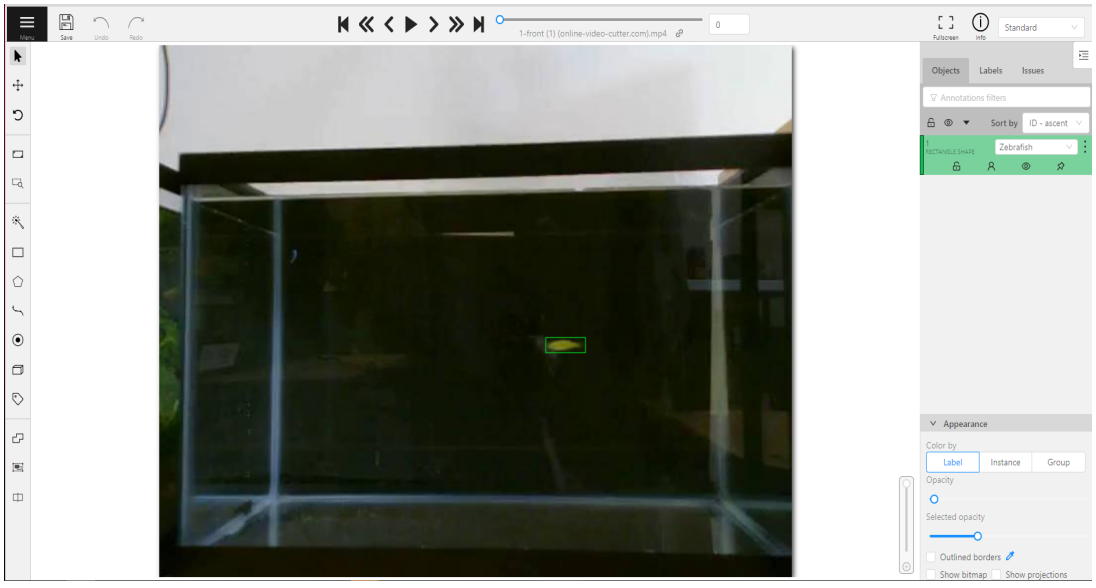

Figure 5. Environment for labelling individual frames of video in CVAT

6.3 Training the Network

After fixing the annotation in all frames of the video, we exported the data and ran the training on the network. At first, we decided to train the network for 5 epochs and a batch size of 8. All other hyperparameters were the default values of the repository. The repository recommends using the largest bath size possible (for the most accurate calculation of gradient in stochastic gradient descent optimization method). The choice of a batch size of 8 in our case was limited by the VRAM size of the GPU (Nvidia GTX 2080 Super), which unlike the GPUs used specifically for machine learning applications has only 8GB of VRAM. When allocating any larger than a batch size of 8, the GPU ran out of memory and aborted the training, thus we opted for the largest number that did not stop the training.
Initially, as a test run, we trained the network on data from the front view for 5 epochs, which took about 40 minutes. However, due to poor performance, we increased that to 80 which took about 3 hours and 6 minutes. A sample detection result is shown in the figure below.

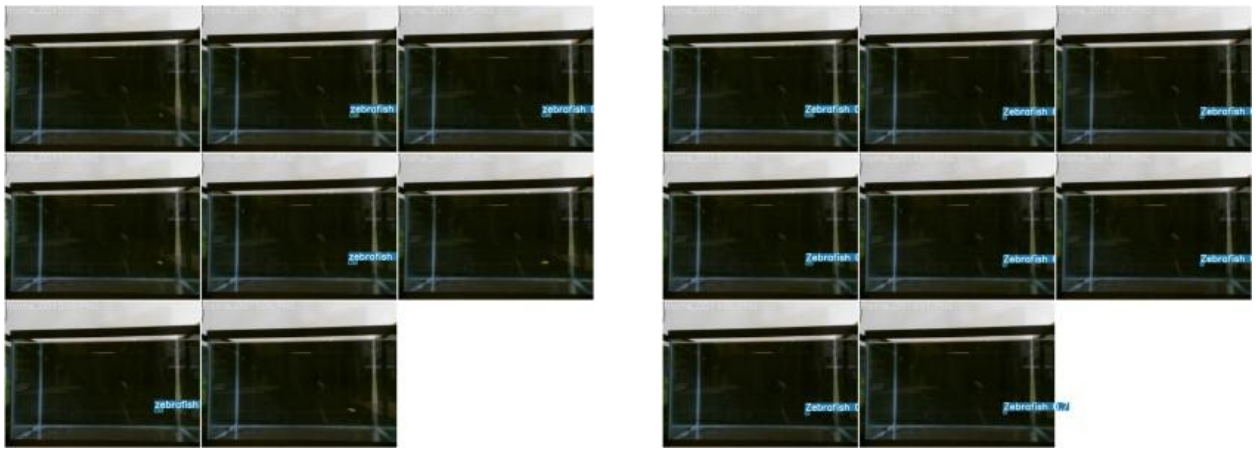

Figure 6. Left: detection results for representative frames after training for 5 epochs. Right: after training for 80 epochs.

Our training results (top) are shown in the figure below and compared to the results provided on the Github repository (bottom). The trend for every performance metric is similar to the reference plot, which indicates that there were no major issues encountered during the training process. Note that, because our dataset content only one class, the plots related to Classification are empty.

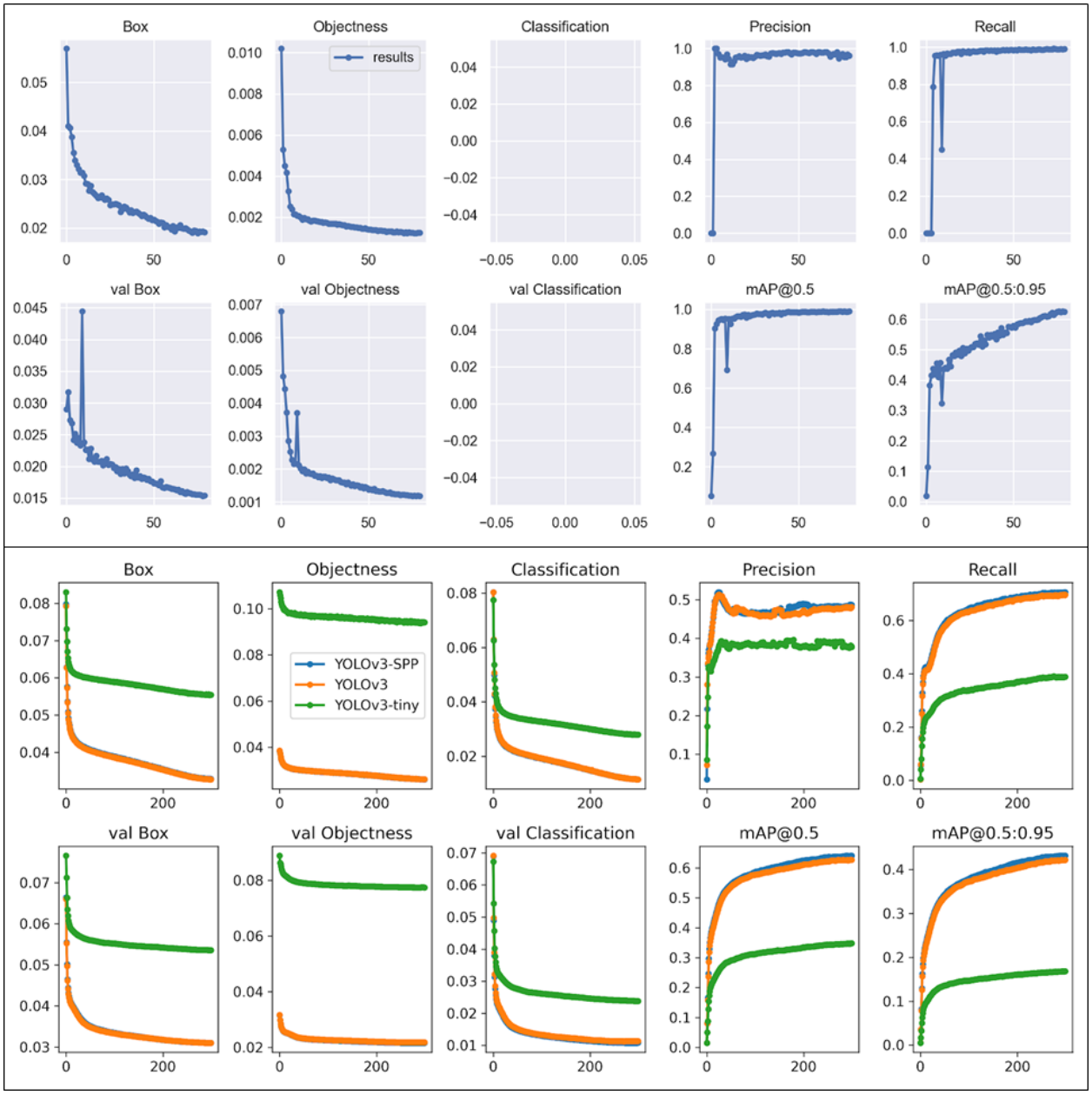

Figure 7. Top: results of training YOLOv3 on our custom dataset. Bottom: results provided by Ultralytics YOLOv3 Github repository.

To achieve optimal results, we trained two separate networks for the front and top views. We also verified that two instances of YOLOv3 can be executed at the same time while achieving better than 30 FPS performance. For the top view, due to time constraints, we trained the network for only 5 epochs (with a batch size of 5). For consistency, for the top view, we used a clip corresponding to the same interval as the front view training clip.

6.4 SORT implementation and tuning

We used the code provided by the authors of SORT (https://github.com/abewley/sort) , which is called after detection is performed on each frame by YOLOv3. SORT provides three tuning parameters:

max_age: the number of frames that the object ID is kept without an associated detection. By setting this value higher, SORT waits for a larger number of frames before discarding the object ID. It can be useful when an object is relatively still but is occluded for a number of frames.

min_hits: the number of associated detections before an object ID is assigned to the object. If the value is lower, an object ID is assigned more quickly to the object.

iou_threshold: the minimum value of Intersection Over Union for the bounding boxes from prediction and detection in order to assign the same object ID to them. If the object moves very quickly or unpredictably (i.e. differs from the constant velocity model used in the Kalman filter), lowering this value could improve detection.

Using the descriptions above and trial-and-error, we set the following values for the parameters, max_age = 100; min_hits = 0; iou_threshold = 0.01

7. Results

For testing YOLOv3 after training it on our custom dataset, we used a different, 30-second portion of the same 10 minutes capture that our training data was trimmed from. This was done for consistency between camera position, and lighting of the scene between training and test data.

Because the goal of this project was improving the tracking performance, here we focus on comparing the performance and abilities of our method with the frame differencing method used before taking this course. Although we could exhaustively quantify the two methods by counting the exact number of frames where the tracking is missing in the frame differencing method, as we will explain in this section, the advantages and reliability of YOLOv3 are obvious enough that render such comparison unnecessary. We should mention that using YOLOv3, in the 900 frame test video of the front view, tracking was lost for only 21 frames (97.7% detection performance); the fish had the same pose in the majority of those frames, meaning that problem can be rectified by diversifying the training data or training for more epochs, or by using data augmentation to create more instance of the edge-case scenario. Because the top-view camera was trained for only 5 epochs, we did not perform a similar calculation for it.

1- The frame differencing method is unable to distinguish between multiple types of fish, while YOLOv3 can easily handle that. Although we trained our network using labeled data of a single species of fish, it is possible to readily expand that to multiple types of fish. The frame

differencing method is only able to pick up movements (optical flow), with no knowledge of the object in the bounding box.

2- In addition to detecting fish movements, the frame differencing method picks up any other movement in the environment such as shadows or the subtle optical flow generated by small vibrations caused by a person walking by the fish tank. This is illustrated in the figure below.

Figure 8. Left: Detection using YOLOv3 when a person walks by the fish tank. Right: Detection in the same frame using frame differencing. Note the detection of the shadow on the wall, and spurious detections due to small vibrations of the fish tank.

We also tried SORT on the detection results with two fish. Although we were not able to capture any videos of more than one fish, we created a video of two fish by overlaying two clips in Adobe Premiere Pro. Note that this changes the pixel values of the video (a shift in the data distribution compared to training data), possibly resulting in worse detection performance compared to an original video. As we have mentioned previously, the performance of SORT depends largely on detection performance, and consecutive missing frames of detection cause SORT to lose track of the fish. Another issue were reflections on the side view of the fish tank. With all that said, the tracking performance was surprisingly robust. In the figure below, we have shown one example of lost tracking due to poor detection performance, which can be easily improved by improving the network's training.

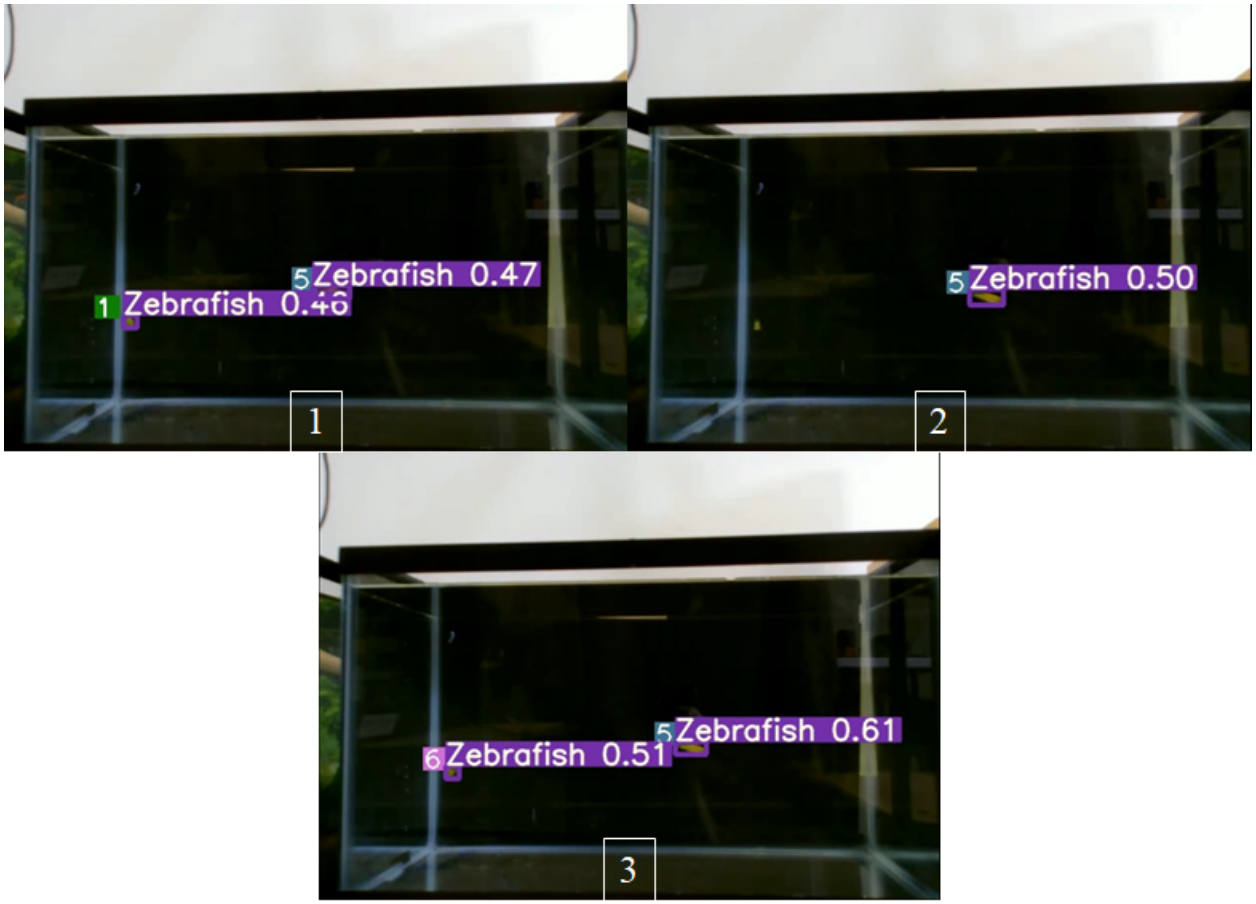

Figure 9. (1) The left zebrafish is tracked with an ID of 1. (2) Detection is lost for 10 frames. (3) Detection is restored, but a new ID (6) is assigned by SORT, likely due to a very low IOU value between the prediction by Kalman filter (based on the velocity 10 frames earlier) and the new detection.

8. Future work

1- Training the network for more epochs, with more data (perhaps using data augmentation techniques to increase the number of samples of edge cases), and for other types of fish. Better training will automatically result in better tracking performance as well.

2- Obtaining the 3D coordinates of each fish using tracking results of the two views.

3- Note that the cameras are not completely orthogonal to the fish tank. Also, note that each plane (side and top view) is trivially defined by four corners of the tank. It should be possible to use the four-point algorithm, or some other transformation method to convert the 3D coordinates obtained using the current setup to those corresponding to the normal views. In general, using more information from camera calibration or the physical setup is a good idea.

4- Adding this system to the other components of FishVR (e.g. real-time visualization of the trajectory of the fish).

9. Conclusion

We presented a system that uses two regular webcams for data collection, YOLOv3 for object detection, and SORT for object tracking. The system can detect and track multiple fish in a fish tank. Our hardware solution is cost-effective and easy to set up. We found that data for YOLOv3 can be labeled reasonably quickly and the network can be trained in a short amount of time on consumer-grade GPUs. We used the output of YOLOv3 as the input to SORT to perform fish tracking. Further, we compared our results to the software system previously used by our client and demonstrated that it is significantly more robust and capable. Finally, we presented the next steps for improving the performance and capabilities of the system. Our progress from the start to finish of this project is shown in the figure below.

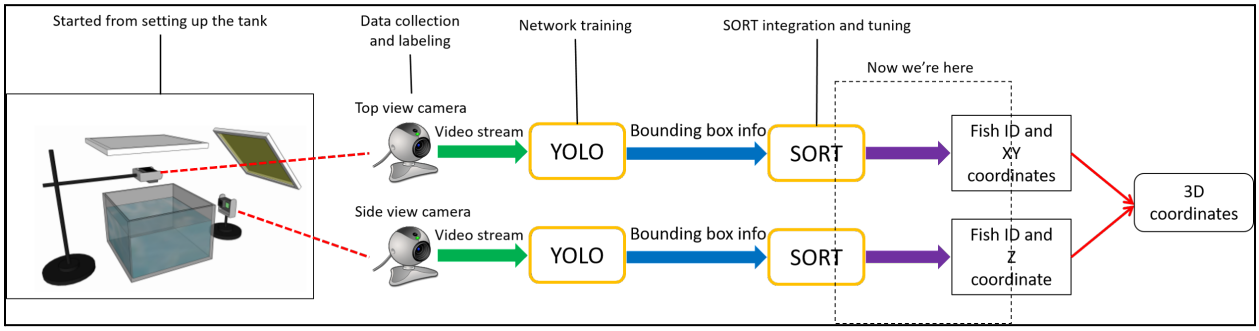

Figure 10: Progress schematic

References

4 Pautsina, A., Císař, P., Štys, D., Terjesen, B. F., & Espmark, Å. M. O. (2015). Infrared reflection system for indoor 3D tracking of fish. Aquacultural Engineering, 69, 7–17. https://doi.org/https://doi.org/10.1016/j.aquaeng.2015.09.002

5 Cheng, S., Zhang, S., Li, L., & Zhang, D. (2018). Water Quality Monitoring Method Based on TLD 3D Fish Tracking and XGBoost. Mathematical Problems in Engineering, 2018, NA. Retrieved from https://link.gale.com/apps/doc/A609853029/AONE?u=maine_orono&sid=AONE&xid=7b6a1dd4

6 Egan, R. J., Bergner, C. L., Hart, P. C., Cachat, J. M., Canavello, P. R., Elegante, M. F., … Kalueff, A. V. (2009). Understanding behavioral and physiological phenotypes of stress and anxiety in zebrafish. Behavioural Brain Research, 205(1), 38–44.

https://doi.org/https://doi.org/10.1016/j.bbr.2009.06.022

7 Blaser, R. E., & Goldsteinholm, K. (2012). Depth preference in zebrafish, Danio rerio: control by surface and substrate cues. Animal Behaviour, 83(4), 953–959.

8 Kuroda, T. (2018). A system for the real-time tracking of operant behavior as an application of 3D camera. Journal of the Experimental Analysis of Behavior, 110(3), 522–544. https://doi.org/10.1002/jeab.471

9 Kays, R., Crofoot, M. C., Jetz, W., & Wikelski, M. (2015). Terrestrial animal tracking as an eye on life and planet. Science, 348(6240), aaa2478. https://doi.org/10.1126/science.aaa2478

10 Martineau, P. R., & Mourrain, P. (2013). Tracking zebrafish larvae in group – Status and perspectives. Methods, 62(3), 292–303. https://doi.org/https://doi.org/10.1016/j.ymeth.2013.05.002

[m1] Pedersen, Malte, et al. "3D-ZeF: A 3D Zebrafish Tracking Benchmark Dataset." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020.

[m2] Redmon, Joseph, and Ali Farhadi. "Yolov3: An incremental improvement." arXiv preprint arXiv:1804.02767 (2018).

[m3] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[m4] Girshick, Ross. "Fast r-cnn." Proceedings of the IEEE international conference on computer vision. 2015.

[m5] Bewley, Alex, et al. "Simple online and realtime tracking." 2016 IEEE International Conference on Image Processing (ICIP). IEEE, 2016.

[m6] Wojke, Nicolai, Alex Bewley, and Dietrich Paulus. "Simple online and realtime tracking with a deep association metric." 2017 IEEE international conference on image processing (ICIP). IEEE, 2017.

[m7] Lin, Tsung-Yi, et al. "Microsoft coco: Common objects in context." European conference on computer vision. Springer, Cham, 2014.