RESEARCH ARTICLE

Using Computer Vision to Quantify Shark-Human Interactions

Aryana Villela Mugnatto

ABSTRACT

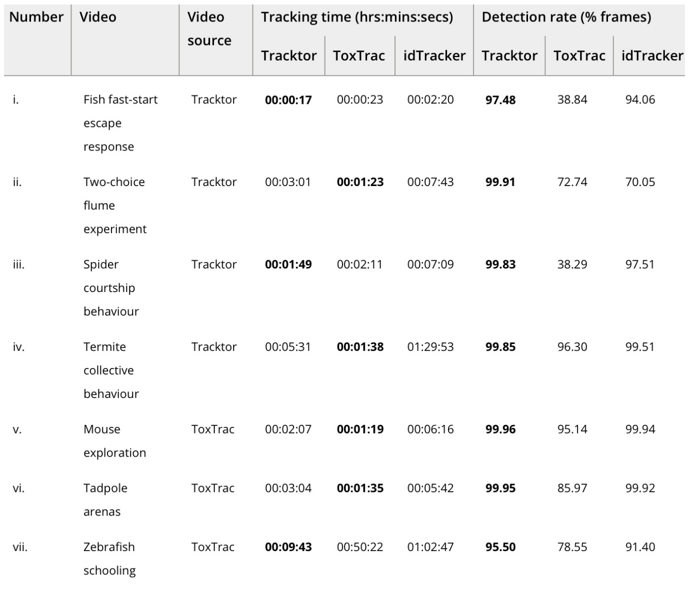
White sharks (Carcharodon carcharias) are a species of shark which can be found in the coastal surface waters of all major oceans. Patterns in their movement and locomotion have been difficult to discover and research due to the challenges that come with tracking large aquatic animals over great distances with precision and accuracy. Our collaborators at CSU Long Beach and the Scripps Institution of Oceanography have been collecting drone video footage of white sharks in their natural habitat along the California coast, including footage of sharks near surfers, bodyboarders or other human beings in order to analyze their behavior and interactions towards people. In order to analyze this data on a large scale, our team has been working to develop novel image processing tools that can accurately and automatically track the movement of sharks and people. Automated movement tracking is essential because of the scale of our data, and offers further benefits such as improved replicability due to the lack of observer bias. Our team developed these tools by building off of existing tracking software, Tracktor, which was designed to track single individuals in a heterogeneous environment or multiple individuals in a homogenous environment. We have been able to significantly improve the tracking software to suit our needs of tracking sharks and people in noisy environments, with significant successes in tracking the sharks and partial successes in tracking the human beings.

INTRO

In order to develop tracking software that was well suited to our needs, we began by selecting a program that we could build off of and improve so that it could successfully track individuals in our videos. Ultimately, we chose a program called Tracktor for a number of reasons.

The program is free and open-source, allowing our team and our collaborators easy access to the source code as well as the improved code. It is a command-based python program, which offers the advantage that it can be easily modified and improved, although some may find it less intuitive than a GUI. However, the program was designed to be very user-friendly even for users with no coding experience, as it can be used in its highly accessible Jupyter Notebook form in which the user only has to specify a few tracking parameters in a designated location. Furthermore, Tracktor is compatible with all operating systems and comes with a user manual detailing the complete installation and use of the software. Tracktor also displays the tracking in real time such that users can visualize the results of their tracking immediately, giving them the option to cancel the task if they see it was not parameterised correctly rather than forcing them to wait until the tracking is complete, and allowing them to visualize the tracking in frames preceding a failure if the program is unable to run for the entirety of the video.

Not only was the usability of Tracktor ideal for our purposes, but its tracking performance was also exemplary. Tracktor was designed to be able to track and maintain the identities of several individuals in a uniform, lab setting, or a single individual in a non-uniform setting. This was crucial because our drone footage contains very noisy, inconsistent backgrounds that many tracking programs cannot overcome. Tracktor significantly outperforms other free, open-source, state-of-the-art tracking software (ToxTrack and idTracker) in terms of detection rate (the percentage of frames in which the individual(s) were reliably detected across the entire video). Tracktor also performs relatively well in terms of tracking time (the time required to process the video), outperforming idTracker for every video tested, and performing approximately as well as ToxTrack. It is also worth noting that the tracking time for each video run by Tracktor was under 10 minutes, while ToxTrack took over 50 minutes to process one particular video (Zebrafish schooling), and idTracker took over an hour to process each of two particular videos (Zebrafish schooling and Termite collective behavior). Below (Figure 1) is a table quantitatively comparing Tracktor, ToxTrack, and idTracker. (Sridhar)



Number	Video	Video source	Tracking time (hrs:mins:secs)			Detection rate (% frames)		
			Tracktor	ToxTrac	idTracker	Tracktor	ToxTrac	idTracker
i.	Fish fast-start escape response	Tracktor	**00:00:17**	00:00:23	00:02:20	**97.48**	38.84	94.06
ii.	Two-choice flume experiment	Tracktor	00:03:01	**00:01:23**	00:07:43	**99.91**	72.74	70.05
iii.	Spider courtship behaviour	Tracktor	**00:01:49**	00:02:11	00:07:09	**99.83**	38.29	97.51
iv.	Termite collective behaviour	Tracktor	00:05:31	**00:01:38**	01:29:53	**99.85**	96.30	99.51
v.	Mouse exploration	ToxTrac	00:02:07	**00:01:19**	00:06:16	**99.96**	95.14	99.94
vi.	Tadpole arenas	ToxTrac	00:03:04	**00:01:35**	00:05:42	**99.95**	85.97	99.92
vii.	Zebrafish schooling	ToxTrac	**00:09:43**	00:50:22	01:02:47	**95.50**	78.55	91.40

Fig. 1. Quantitative comparison of Tracktor, ToxTrack, and idTracker

METHODS

Research

After selecting an existing program, the next step in developing our tracking software involved testing out Tracktor before making any changes to the software on some of our drone videos in order to assess the quality of the tracking in our videos and to plan for how to improve the program for our specific needs. Before testing out the program on our videos, however, we analyzed the source code in order to better understand which tools and methods Tracktor used to attempt to track objects. Our analysis of the code revealed that the program relied on three key algorithms to track objects: adaptive thresholding, the K-means algorithm, and the Hungarian algorithm.

Adaptive thresholding is the first step in the process that allows the program to separate the foreground of the video from the background, in an attempt to isolate the objects we wish to track. The method is an off-the-shelf openCV method cv.adaptiveThreshold and functions by applying a threshold value to each pixel, and assigning the pixel to the foreground if it is above the threshold and to the background if it is below the threshold. Adaptive thresholding is better suited to our needs than global thresholding, which uses one threshold value for the entire image, while adaptive thresholding determines the threshold for a pixel based on a small region around it. This is because the dynamic threshold value accounts for different lighting conditions within one image, which is relevant to our videos as they do not occur in a lab setting with uniform lighting. Adaptive thresholding is also better suited for our needs than background subtraction, which also attempts to isolate the foreground of a video from the background, but does so by assigning moving parts of the video to the foreground and stagnant parts of the video to the background. This would not work for our videos, which have a dynamic, moving ocean as the background to our target objects.

The next step in the tracking process uses the K-means clustering algorithm. K-means clustering is an unsupervised learning algorithm that aims to partition unlabeled data into K groups or clusters based on feature similarity. The algorithm is given a specified K and then randomly selects K data points and assigns each point to a cluster. It then computes cluster centroids and iteratively optimizes the centroids until the sum of the squared distance between the data points and their centroids is minimized. Tracktor uses this algorithm to resolve instances in which the number of objects detected in a frame does not match the number of objects expected, K. The program then creates K clusters of objects and assigns the tracked identity to the centroid of each cluster.

The final step in the tracking process uses the Hungarian algorithm. The Hungarian Algorithm takes the coordinates of the objects in a given frame as well as in the previous frame and minimizes a cost function between points in subsequent frames in order to assign and maintain identities. The algorithm works by computing a matrix for each frame that is filled with the distances between the previous frame (rows) and the current frame (columns) for all objects being tracked, and then minimizes the distance between an object in the previous frame and the same object in the current frame. Tracktor uses this algorithm when it is attempting to track multiple objects in order to maintain their identities from frame to frame.

Testing

Once we understood the underlying methods used by Tracktor to track objects and maintain their identities, we could begin the process of running our own videos through Tracktor to see how it would perform. Initially, Tracktor could not even process our videos to begin the tracking process. By comparing our videos to the sample videos provided by Tracktor, we were able to identify features shared by all of the sample videos that Tracktor was able to process that differed from our videos. We then converted the codec, FPS, and size of the video to match those of the sample videos, which resolved the problem and allowed Tracktor to process our videos.

Once Tracktor was able to process our videos, the real-time tracking display allowed us to see how the program was tracking objects in our videos frame-by-frame. When the parameters were set such that the program expected to track a single object, the tracker did not seem to track a specific target and instead seemed to move somewhat randomly near the center of each frame. We quickly realized that this was due to the fact that the adaptive thresholding step of the program was assigning many objects to the foreground, and the K-means algorithm (with K = 1) was placing the tracker near the center of the cluster of objects. After we changed the parameters such that the program expected to track five moving objects instead of one (K = 5), the program performed much better in terms of placing a tracker on the shark for much of the duration of the video. The difference in performance can be seen in the tracking plots output by the program (Figure 3). However, Tracktor also tracked multiple other objects in the video, namely waves breaking in the ocean and on shore, and the program had no way of distinguishing the shark from these other objects.

Tracktor allows users to easily change several parameters that affect which objects are identified and tracked. One set of parameters that could help the program distinguish between the sharks and humans and other objects is minArea and maxArea, which allows the user to specify upper and lower boundaries for the pixel area of the object being tracked. This allowed us to filter out any object that was far larger or smaller than the sharks or humans, but there were still many waves and other foreground objects identified by the adaptive thresholding methods that were approximately the same size as the sharks and humans. The user can also vary blockSize and offset, which are parameters of the adaptive threshold method that determine the size of the region around a pixel from which a threshold is determined and the constant that is subtracted from the calculated threshold, respectively. We systematically varied these parameters and and compared the binary images that resulted from the parameterized thresholding (Figure 2), but found little to no difference in the tracking output of the program.

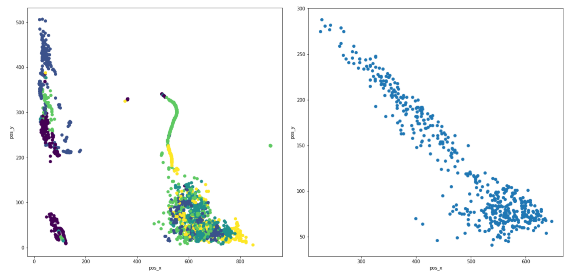


Fig. 2. Partial success tracking shark shown in green and yellow path with five trackers (left), and failure with one tracker (right)

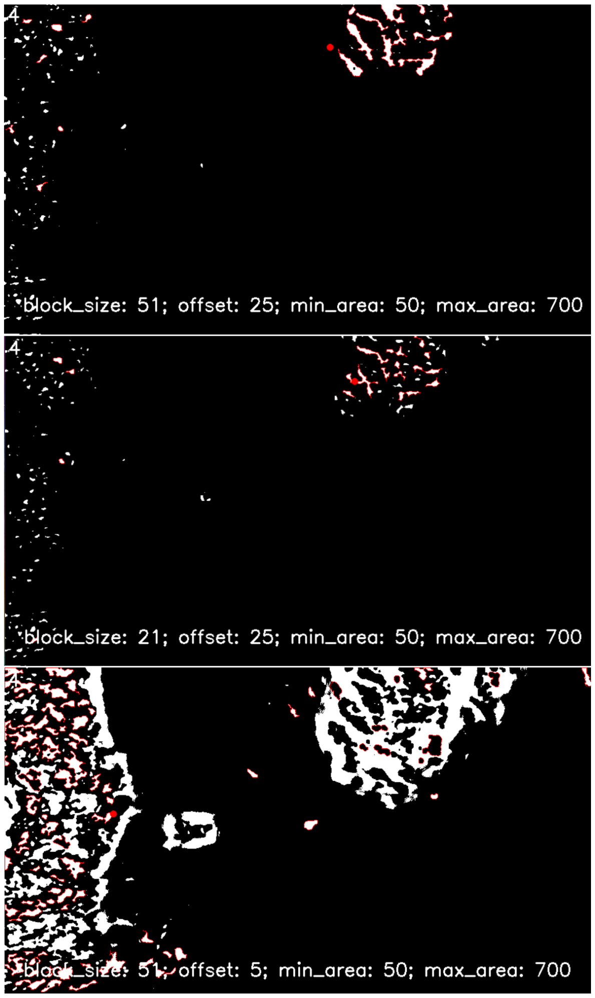

block_size: 51; offset: 25; min_area: 50; max_area: 700
block_size: 21; offset: 25; min_area: 50; max_area: 700
block_size: 51; offset: 5; min_area: 50; max_area: 700

Fig. 3. Binary images resulting from variations in blockSize and offset

Development

After fine-tuning the existing features of Tracktor to best suit our videos, it became clear that novel features needed to be implemented in order to effectively distinguish the shark and human targets from similarly sized objects being identified and tracked by the program. In order to do so, we needed to identify certain properties that were consistently present in the target objects that we were attempting to track, and were consistently absent from all other objects identified by the thresholding step of the tracking process.

One property of trackable objects that we considered when attempting to distinguish between sharks, humans, and other objects was shape, particularly the shape of the contours in the binary image produced by the adaptive thresholding step. While the contours of the shark generated by the thresholding step of the program contained specific features that were roughly consistent throughout the video (fins, tail, etc.) there was no simple or obvious method of identifying these features in order to filter out other objects beyond filtering based on size, and we decided to move on and consider other properties of the objects being tracked.

The next property we considered was the intensity values of the pixels in each object. Our first step towards better understanding the color distribution of the video was to implement a method that would output a histogram of the intensity distribution of the greyscale image that was inputted into the adaptive thresholding method. Because most of the objects that we did not wish to track that were identified by the adaptive thresholding method were breaking waves that were approximately white in color, we were hoping that the associated pixels would be significantly higher in intensity, resulting in a bimodal histogram of higher intensity pixels and lower intensity pixels, with a lower proportion of medium-intensity pixels. Unfortunately this was not the case, and there was no clear distinction between the lighter and darker portions of the image, with the histogram revealing very few pixels at either extreme (Figure 4).

We then decided to investigate the pixel color values of the three-channel color images rather than the pixel intensity values of the single-channel greyscale image in hopes of extracting more useful information that could be used to distinguish between our target objects and the other objects being tracked. We implemented a method that would output the average BGR color values of every object that was identified by the adaptive thresholding step of the program in an attempt to identify unique properties of the BGR values associated with the shark or human objects. Unfortunately, we were unable to identify a range of BGR values that was unique to the target objects we intended to track.

We then began to output the average HSV color values of every object identified instead of BGR values to compare the hue, saturation, and value of our target objects to all other objects being tracked. We found that, while the approximate hue, saturation, and value of the shark object were not unique individually, the combination of all three were not found in any other objects within a fairly large range (+- 20 for hue, saturation, and value, which range from 0 to 180, 0 to 255, and 0 to 255, respectively). Following this discovery, we decided to include HSV value as one of the features that we would base our tracking on.

We then developed two versions of the tracking software that would consider HSV value using two different approaches, HSV filtering and HSV thresholding. The HSV filtering version functioned very similarly to the original program. It began the same way, with a greyscale version of each frame being inputted into the adaptive thresholding method, resulting in a binary image including white, trackable foreground objects and a black background. As before, the program would then filter out the objects outside of the area constraints specified by the user; however, this version of the program contained an additional step in which the program would also filter out the objects outside of the HSV range specified by the user. Initially, the HSV was entered directly into the code by the user, requiring the user to know the HSV value for which to center the HSV range. However, to increase usability and generalizability, we implemented a method that displayed the first frame to the user before the process began, allowing the user to click on the object they intended to track, thereby extracting the HSV color information about the pixel they clicked on and using that value as the center of the HSV range that would then be used to filter the foreground objects.

The second version of the tracking software we developed used an HSV thresholding approach. This version was considerably different from the original software as we entirely replaced the adaptive thresholding step with a newly implemented HSV thresholding step. We used the same display-and-click method of

HSV color extraction as we did in the HSV filtering approach in order to provide a basis for our HSV thresholding. Instead of inputting a greyscale image into the adaptive thresholding method and thresholding based on intensity, we inputted the full color image into openCV's color thresholding method, inRange(), which would assign pixels that are within the specified range to the foreground, and all other pixels to the background. This was meant to ensure that all of the objects initially identified by the thresholding step were already within our desired color range such that no color-based filtering would be necessary afterwards. The objects generated by this method were somewhat more fragmented than the objects generated by the adaptive thresholding method, with gaps or holes in the objects due to single pixels being outside the specified color range, and single pixels within the color range that are outside of the objects being assigned to the foreground. To resolve this problem, we implemented a simple dilation and erosion step that would fill the gaps within objects via dilation and erode single pixels outside of the objects.

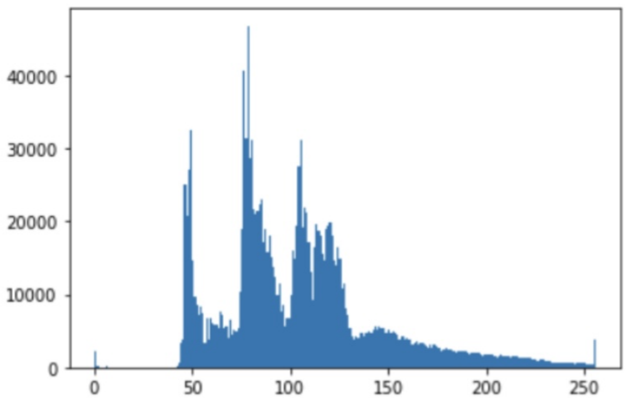


Fig. 4. Histogram of intensity values in a single frame

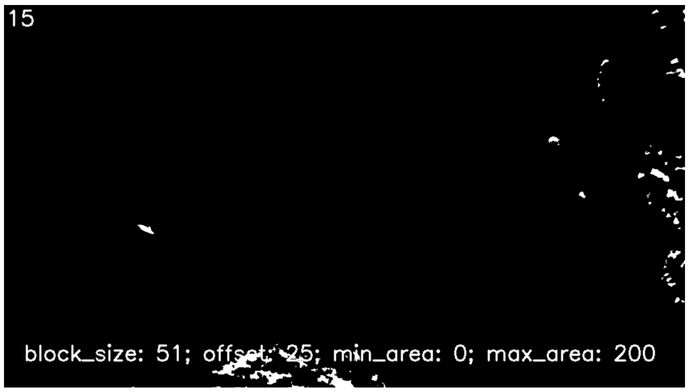
RESULTS

The results of the HSV filtering and HSV thresholding versions of the software showed partial success in our goal of tracking all sharks and humans in a video for the entire duration of the video. For both the HSV filtering approach and the HSV thresholding approach, the shark was tracked for the majority of the time for the majority of our high quality drone footage videos. However, when we first attempted to track humans using the HSV filtering, the tracking was unsuccessful at tracking only the humans for significant periods of the video, and would either track additional objects besides the humans that we intended to track, or would fail to track anything at all. It became clear that further analysis was necessary in order to identify the properties of the humans that made them so much more difficult to track than the sharks using this approach.

We began our investigation by analyzing the binary image that resulted from the adaptive thresholding step of the process (Figure 5). Foreground objects were present at the location of the humans in the video, revealing that the adaptive thresholding step was, in fact, identifying the humans as trackable objects. To get a clearer understanding of what exactly the adaptive thresholding step was detecting, we overlay the contours of the adaptive thresholding foreground onto the color image of each frame, effectively outlining the objects that were identified by the adaptive thresholding. We found that only certain parts of the humans were being assigned to the foreground, namely the dark wetsuit of the person and the shadow of the person and the surfboard or bodyboard. This was likely because the extreme color contrasts within what we would consider to be the human object (including the person themselves and the surfboard or bodyboard) resulted in thresholding within said object rather than thresholding of the object itself from the rest of the background.

This was crucial information as we had been primarily selecting pixels from the brightly colored surfboards and bodyboards, as they had the most unique HSV values relative to the rest of the image, but these pixels had been assigned to the background during the adaptive thresholding step, and could not be tracked as foreground objects. We attempted to instead track the shadows of the humans, as the shadows were the part of the humans that were identified as foreground objects, and although we were successful in consistently tracking the human shadows, we were unsuccessful in distinguishing these shadows from other shadows in the image, as they were not unique in HSV color value. Because of the K-means clustering step of the program, this resulted in the tracker appearing somewhere between the shadows that were identified, and did not follow the humans closely (Figure 6).

The problem of the uniquely colored surfboards and bodyboards not being assigned to the foreground was resolved with the HSV thresholding version of the program, which would allow us to only assign pixels within our specified HSV color range to the foreground. This approach worked well within the first few (20 - 30) frames of the video; however, the program would eventually stop assigning any objects to the foreground, presumably because there existed no objects within the color range. We attempted to resolve this problem by creating an adaptive version of HSV thresholding, in which the center of the HSV range would be updated after every frame by taking the average color of the foreground of the previous frame. We hoped that this would shift the HSV range to reflect changes in lighting conditions that affected the HSV value of the pixels of the surfboard or bodyboard. Unfortunately, this adaptive version of the HSV thresholding approach did not improve the tracking, and in fact, the program stopped assigning any objects to the foreground even earlier, stopping after about ten frames.



block_size: 51; offset: 25; min_area: 0; max_area: 200

Fig. 5. Resulting binary image after adaptive thresholding

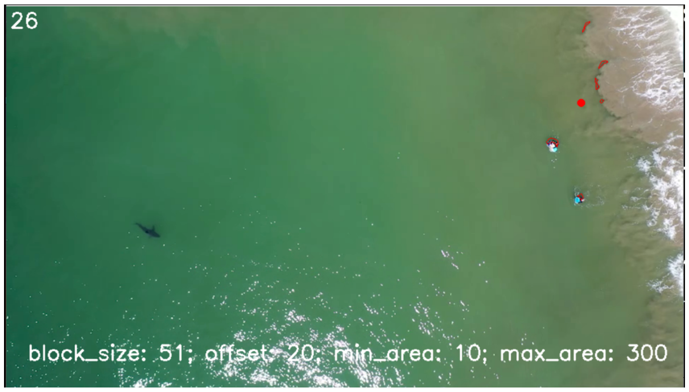


Fig. 6. Resuting tracker after filtering based on color of surfers' shadows

DISCUSSION

There is still much that can be done to improve the performance of our tracking software. While the program tracks the shark quite successfully, no version of our program is able to track any humans for an extensive period of time. A thorough investigation of how the HSV values of the surfboards and bodyboards change over the course of each video, as well as of how the HSV range changes as it is updated after each frame, could result in new insights that that may allow the HSV thresholding version of the program to be improved to the point of successfully tracking the humans. Further analyses of unique properties of the humans we wish to track could also result in novel methods of identifying and distinguishing them from other objects; for instance, the combination of various highly contrasting colors that is uniquely associated with our human targets may be taken advantage of.

We have also started exploring entirely different approaches to meeting our goals without using Tracktor, particularly machine learning approaches. We have started the process of video annotation for deep learning using the website Supervise.ly, which allows users to easily and efficiently create data sets for which to train neural networks on. While this approach seems promising, we have only just begun the time-consuming process of annotating our existing videos to train a neural network for future videos, so the outcome of this approach is currently inconclusive. However, there is much potential for success, especially because our drone footage videos and the objects we wish to track within them are all relatively similar in nature, so a thoroughly trained neural network may very well be able to successfully track sharks and humans for extensive durations of our videos.

REFERENCES

Sridhar, Vivek Hari;, Roche, Dominique G.; Gingins, Simon; et al. (2019). "Tracktor: Image-Based Automated Tracking of Animal Movement and Behaviour." Besjournals, John Wiley amp; Sons, Ltd, 5 Mar. 2019, besjournals.onlinelibrary.wiley.com/doi/full/10.1111/2041-210X.13166.