AUTOMATIC FISH TRACKING: KEEPING TRACK OF WHO'S WHO

by

Ari Spraggins

A Thesis Submitted to the Faculty of

The Wilkes Honors College

in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Liberal Arts and Sciences

with a Concentration in Physics

Wilkes Honors College of

Florida Atlantic University

Jupiter, Florida

May 2021

AUTOMATIC FISH TRACKING: KEEPING TRACK OF WHO'S WHO

by

Ari Spraggins

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Yaouen Fily, and has been approved by the members of their supervisory committee. It was submitted to the faculty of the Harriet L. Wilkes Honors College and was accepted in partial fulfillment of the requirements for the degree of Bachelor of Science in Liberal Arts and Sciences.

SUPERVISORY COMMITTEE:

Dr. Yaouen Fily

[second reader]

Dean Justin Perry, Harriet L. Wilkes Honors College

Date

Abstract

Author:	Ari Spraggins
Title:	Automatic Fish Tracking: Keeping Track of Who's Who
Institution:	Harriet L. Wilkes Honors College, Florida Atlantic University
Thesis Advisor:	Dr. Yaouen Fily
Degree:	Bachelor of Science in Liberal Arts and Sciences
Concentration:	Physics
Year:	2021

Automatic video tracking has had a major impact on animal behavior studies. One of the challenges of this technique is keeping track of the identities of the fish, especially when they swim together and exchange positions. In this project we use the python programming language to address this problem for groups of fish. The video data comes from schooling assays performed at FAU's Cavefish Trilab (Dr. Keene, Dr. Duboue, and Dr. Kowalko). The method is inspired by the idTracker animal tracking software: we track patterns of brightness as a visual identifier of each fish which we then use to detect when the fish swap places.

Contents

List of Figures

1 Introduction

1.1 Tracking Animal Behaviour

While visually pleasing, schooling is a rather challenging topic that has long intrigued animal scientists. In order to quantify the animals' behavior, a vast quantity of positional data is needed; more than can be collected by hand. Advances in computer vision technology now make this type of data accessible through computerized video-tracking. Beyond schooling, this allows to quantify a wide variety of behaviors in a very accurate manner.

This thesis focuses on tracking the motion of Mexican Tetra fish (*Astyanax mexicanus*). One of the quirks of the Mexican Tetra species is that some of its populations have been living in underground caves for about a million years. There, they have evolved a number of behavioral differences with the populations living on the surface, including the loss of schooling: whereas surface populations of *A. mexicanus* school, cave-adapted ones do not [**?**].

The data used in this thesis comes from experiments performed on campus in the labs of Dr. Keene, Dr. Duboué, and Dr. Kowalko, collectively known as the Cavefish Trilab. The tracking software that the project aims to improve on was developed by Dr. Fily's group for the Cavefish Trilab [**?**, **?**]. The current version of that software has trouble maintaining the identities of the fish throughout an experiment. The purpose of this work is to fix that issue.

1.2 Nature of the problem

Before discussing how to maintain the identity of the fish as we track them, we must first talk a little about how the software locates them. Throughout the thesis **we focus on two-fish experiments**. In each frame of the video, the software identifies regions that are darker than their surrounding. After filtering out dark regions whose size of aspect ratio is inconsistent with a fish, each remaining dark spot is interpreted as a fish. This works well for determining the positions of the fish at any given moment, but the identities of the fish can swap (fish 1 becomes fish 2 and vice versa) at any time.

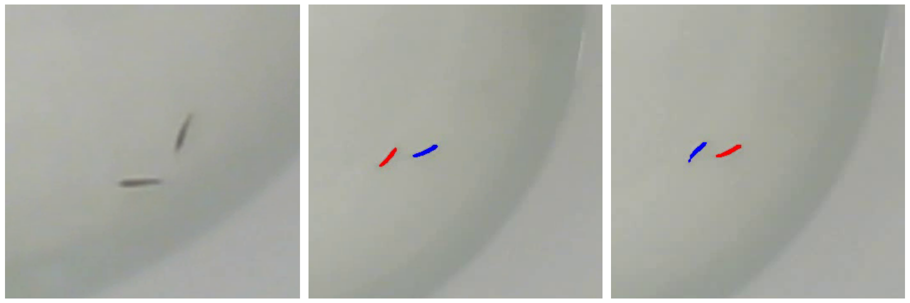


Figure 1: *Left*: Close-up of two fish in their tank. The tank is made of white plastic. Each fish appears as a dark spot. *Middle and Right*: Two consecutive frames of a two-fish video. Fish 1 is highlighted in blue. Fish 2 is highlighted in red. Basic dark spot detection makes no attempt to maintain the color ID of the fish, i.e., the colors can swap at any time.

Many of those identity swaps can be fixed by analyzing the distance traveled by each fish. The videos are shot at 30 frames per second, so fish do not move much from one frame to the next. Therefore, the correct ID of a dark spot can often be obtained by matching each dark spot in the frame to the closest dark spot in the previous frame.

However, this does not always work. Issues occur in two scenarios: when the fish get close enough that the program thinks there is only one fish (figure 2), and when the fish move fast enough that they are closer to the other's previous position then their previous position. The first of these issues is the easier of the two to check, since it is fairly trivial to check for regions in which only one fish is reported (please see Appendix A). While it is easy enough to check for the first of these issues, the second is a little tougher. To check for errors in these cases, the distance of each fish from its previous position must be compared to the distance from each fish to the other fishes previous position (Appendix B). If the software mistakes the positions of the fish, which we will refer to as a swap, it will look like the fish traveled longer than it should have.

If the fish are near each other and moving fast, then a fish may closer to the previous position of another fish than to its own previous position. In that case, the distance-based method will swap the two fish's identities. If the fish are even closer, or if one is over or under another, the algorithm may only detect a single large dark spot. We call this an overlap event. An example is shown in Figure 2. Eventually, the fish separate and the algorithm detects them as distinct dark spots again, but the fish identities are lost in the process and

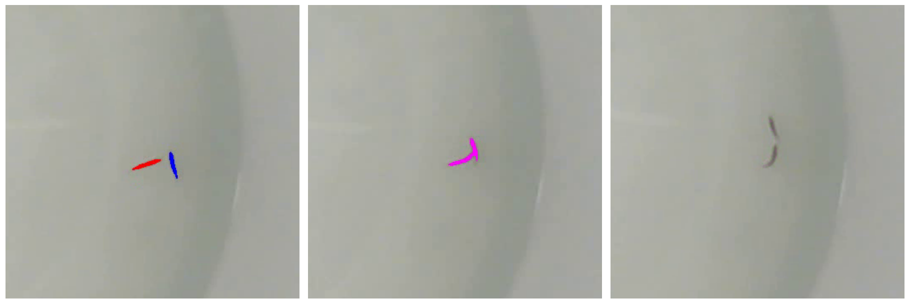


Figure 2: Overlap event. *Left*: Before the overlap, fish 1 (blue) is on the right and fish 2 (red) is on the left. *Middle*: During the overlap, the dark spot detection algorithm only detects one object (purple). The identities of the fish are meaningless because the algorithm thinks they are at the exact same place. *Right*: After the overlap, the fish identities cannot be recovered by analyzing the distance traveled.

1.3 Previous work

Video-tracking software designed with multi-animal experiments in mind typically handle the first type of identity swap described above – the ones that are fixable by analyzing the distance traveled between frames. Swaps caused by overlap situations are trickier. Some older video-tracking software offer the possibility to fix the animals' tracks by hand after running the automatic tracking. A popular example is the commercial video-tracking software Ethovision, made by Noldus. The downside of this approach is that it is slower. In some cases, fixing all identity swaps by hand can become the most time-consuming part of the project.

In recent years, several new software have come out to address this issue using various strategies.

The one this thesis is most heavily inspired from is idTracker [**?**]. The idea is that each fish has a unique pattern of brighter and darker spots on its body, which can be used to recognize the fish even after loosing its track for a while. To quantify those patterns, the program measures spatial brightness correlations over each fish's body and compare them with previous frames to decide whether an identity swap has occurred. A more recent version of this software, known as idTracker.ai, uses machine learning methods to perform this task.

More recently, TRex [**?**] provides a similar workflow but uses deep-learning methods to identify a visual pattern unique to each fish.

DeepLabCut [**?**] also uses deep-learning methods, however it solves a more ambitious problem: tracking multiple points of interest on each animal in order to quantify their posture as well as their position. The trade-off here is that the user has to manually identify the points of interest in several frames across the video, which is a bit more labor-intensive.

1.4 Roadmap

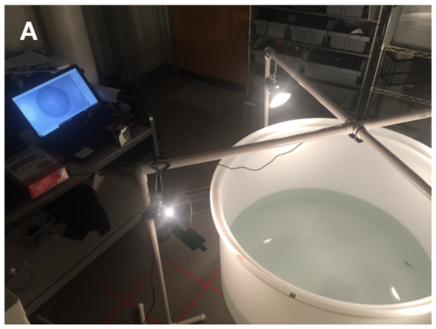
The goal of this project is to draw from the original idTracker approach to help fix identity swaps in a pre-existing video-tracking code designed for use in Cavefish Trilab: cvtracer [**?**, **?**].

[Give a brief overview what's covered in each section of the thesis]

2 Methods

2.1 Video Collection

While any actual experimentation on the fish is beyond the scope of this thesis, it is still useful to describe the process of gathering data. The basic setup of the labs we are taking data from is a tank with two fishes in it and a camera trained on them, as seen below. This setup produces a video for us to use, of which an example frame is shown below.



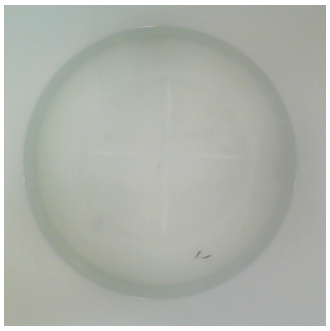


Figure 3: *Left*: The tank setup. *Right*: An example frame of the video.

2.2 Current Tracking

To track the fish, we must first feed the video captured from this setup to an analysis program, in our case cvtracer [**?**, **?**], which has been especially created for use by the Jupiter Trilab. The tracker works taking the tank as a constant background, and noting that the fish are the only dark spots on the tank. It then returns an array for each of the fish containing a list of the fish's pixels and those pixel's colors. Once we have the fish saved in a format that we can analyze, we are ready to start working on the tracking of the fish. Since each the way we store the data gives each fish an identifier, we then can use those identifiers to analyse the fish.

2.3 Distance Based Unswapping

The first and simplest round of identity unswapping works by minimizing the total distance traveled by the fish between the previous frame and the current one. Figure 4 illustrates the way the method works for two fish. The semi-transparent fish indicate each fish's position in the previous frame. In the left panel, identities are maintained, and the distances traveled (double arrows) are small. In the right panel, an identity swap has occurred: in the new frame, the red fish has become blue and vice versa. Because of this swap, each fish appears to have traveled much more distance (double arrows) than it actually has.

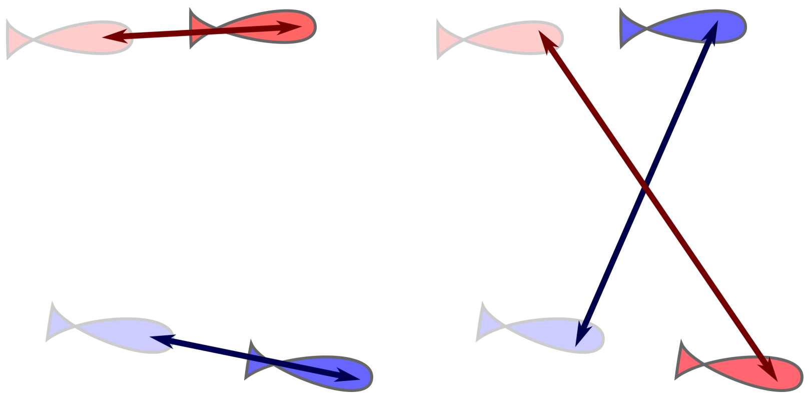


Figure 4: Distance traveled by the fish between two frames when identities are maintained (left panel) and when identities are swapped (right panel).

In order to determine whether a swap has occurred, we compute the following distance matrix, which contains all four of the distances shown in the figure:

$$\begin{pmatrix} \begin{array}{l}\text{distance from fish 1 in previous}\\ \text{frame to fish 1 in current frame}\end{array} & \begin{array}{l}\text{distance from fish 1 in previous}\\ \text{frame to fish 2 in current frame}\end{array} \\ \\ \begin{array}{l}\text{distance from fish 2 in previous}\\ \text{frame to fish 1 in current frame}\end{array} & \begin{array}{l}\text{distance from fish 2 in previous}\\ \text{frame to fish 2 in current frame}\end{array} \end{pmatrix}$$

The sum of the diagonal terms (previous-1-to-current-1 and previous-2-to-current-2) represents the total distance traveled by both fish since the previous frame assuming identities were maintained. The sum of the two off-diagonal terms (previous-1-to-current-2 and previous-2-to-current-1) represents the total distance traveled by both fish since the previous frame assuming identities were swapped. If the latter is smaller than the former, then we conclude the identities were swapped.

2.4 Histogram Based Unswapping

While it would be ideal if the distance based approach could be used for every instance of a swap, unfortunately a limitation of the approach is that it only works in frame groups where the program detects two fish. As such, a second approach is needed for one fish regions. The approach that was decided on for these sections was to take a identifier that was unique to each fish, and compare them against each other on a frame by frame basis. The identifier that we chose was the histogram created by taking the brightness of each fish (Appendix D), as proposed by idTracker[**?**], which has the advantage of being both one of the most unique things that it is possible to analyze about the fish and being easy to compare given that we can take the euclidean distance between histograms. To create these histograms, we are using numpy's histogram2d, which is ideal for our use case since it can be used directly without any other setup on our part besides determining bins, which we will do later in section 3.4.

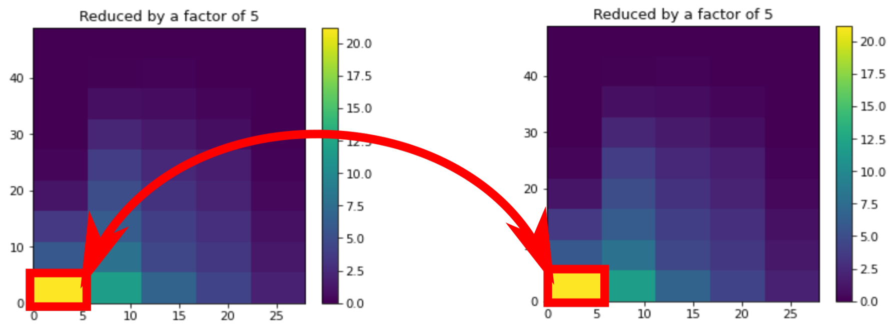


Figure 5: The histograms

2.5 A basic overview of Analysing Histograms

As the histogram based unswapping method only works in time frames of the video in which there are two fish detected, the regions with only one fish detected must be identified so that the correct unswapping method can be used on these regions (see figure 6). The approach that was used for determining these overlapping regions was to comb through the list of the fish in each frame for the amount of fish that the program detected. This was done by using the fact that the program returns a single data point for each fish detected one one of the data's layers, which made determining the amount of fish detected a simple matter of iteration (see Appendix A).

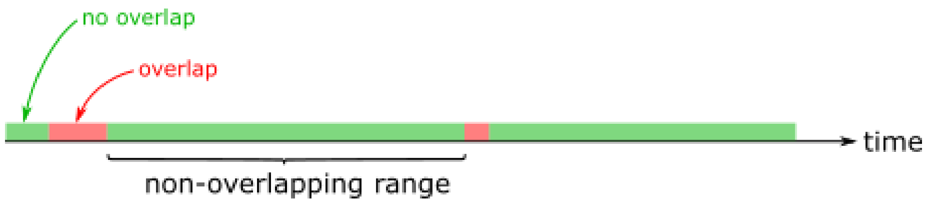


Figure 6: The nonoverlapping ranges

Once the overlapping ranges were detected, to reduce variability in our answers we averaged the regions over which the histograms were taken. By taking the histograms at the beginning and end of the overlapping range, we are able to both reduce the amount of operations performed on each overlap to just one comparison instead of one per frame, and for us to gain more confidence in our answer due to it being an average of multiple data points.

2.6 Determining Accuracy

When determining how accurate the program is at unswapping the fish, there must be a baseline that the data is compared to. For the purposes of checking the data we had generated, a short segment of video was analysed by hand to create a list of each of the fish and weather or not a swap had happened. This allows us to use a known good basis for comparison, and by comparing the numbers of swaps recorded and their positions, we are able to generate a metric for how accurate our process is.

3 Results

3.1 Distance Based Unswapping

One thing that is constant about the fish, is their size, which makes the analysis simpler in several regards. Since the length of the fish never goes over 30 pixels, there are two assumptions that this leads us to: that we can set a minimum distance that any distance between the fish can't be less then, and that it allows us to set a size on the histograms that is constant. This allows us to standardise a bin size by taking this value as the size along one axis and the other axis can be taken by determining the range of values that we can reasonable expect to occur,which can be found by taking sample histograms, and setting it as the maximum of that value.

3.2 Histogram Based Unswapping and Overlap Identification and Reduction

The output from the histograms that are produced by the program are shown below. While these are not easily human-readable, they provide an easy waay to perform computations on the fishes.

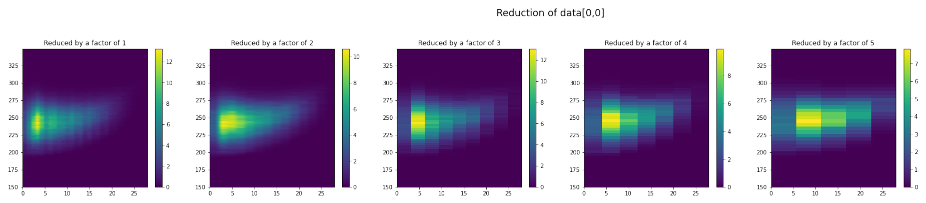


Figure 7: An example histogram

3.3 Accuracy

Without any tuning, the error rate is only 19%. This is probably due to the fish being too uniform, as once are able to tune the process, we can expect a slightly more accurate result.

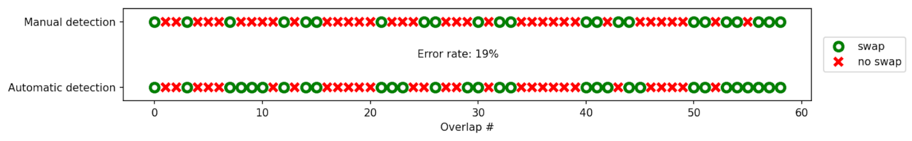


Figure 8: The error of the untuned process

3.4 Bin Reduction

One thing that was also tested was what would happen if less bins were used on the histograms used for analysis, both to see how it affected accuracy and performance. What was found was that there was no significant change in the accuracy of the data when this operation was performed. The most difference in accuracy between the reduced and nonreduced datasets that we were able to observe was 13%, with that being an outlier considering most reductions were under 9%, with the vast majority being under 6%.

4 Conclusion

One problem faced by scientists working on animal tracking tasks is keeping track of which animal is which. To solve this problem for the Jupiter Trilab, we aimed to created a software that detects fish based on a unique identifier. To do this, first we looked at the two types of identity swaps that occur in the original software: a first type that can be fixed by analysing the distance traveled by the fish from frame to frame, and a second type where the fish are so close to each other that the image recognition software only sees one fish. For the second of these swaps, we took the visual identifier of the brightness histogram of each fish and used that to determine the fish's identities. These two methods ended up as being rather successful, given that error checking the data against manual error correction led to a reported error rate of 19%, before any tuning of the parameters. Because the datasets that are being analysed by this program have remarkable amounts of uniformity, it should be fairly simple to drastically reduce this error rate. In addition to the software tuning that can be done for more accuracy, and while not something that can be done inside the scope of our current project, modifications can also be made to the data gathering setup, such as increasing the resolution of the video to increase the amount of data that we have to work with, and changing the types of fish given that Mexican Tetra of different ages have different colourations, though that would be something that would have to be worked around for the final release.

A Locating Nonoverlapping Ranges

```python
i2=0
nonOverlappingRange=[]
while i2<len(fish):
    i1=i2
    while i1<len(fish) and len(fish[i1])!=2:
        i1+=1
    i2=i1
    while i2 < len(fish) and len(fish[i2])==2:
        #find the first overlapping index
        i2+=1
    nonOverlappingRange.append([i1,i2])
print(nonOverlappingRange)
```

B Distance Based Swap Checking

```python
def swapStatus(pos,i):
    '''
    Detect swaps between consecutive frames based on proximity.

    Input:
        pos:Postionts. Array with shape (Nframes,Nfish,Ndimensions),
        i: Frame index. Int.

    Output:
        Int. 0 if no swaps, 1 if swapped, 2 if overlapping.
    '''
    nFish=pos.shape[1] #Number of fish
    distanceMatrix=[np.linalg.norm(pos[i+1][0]-pos[i][0]),
                    np.linalg.norm(pos[i+1][1]-pos[i][1]),
                    np.linalg.norm(pos[i+1][0]-pos[i][1]),
                    np.linalg.norm(pos[i+1][1]-pos[i][0])]
    swapCriteron=(distanceMatrix[0]+distanceMatrix[1])-(distanceMatrix[2]+distanceMatrix[3])
    if abs(swapCriteron)<1e-10:
        return 2 #Overlapping
    elif swapCriteron>0:
        return 1 #Swapped
    elif swapCriteron<0:
        return 0 #Normal
    else:
        return -1
```

C Data Shape

The data is stored in an array of shape [frame][fish][xpixels,ypixels][color]

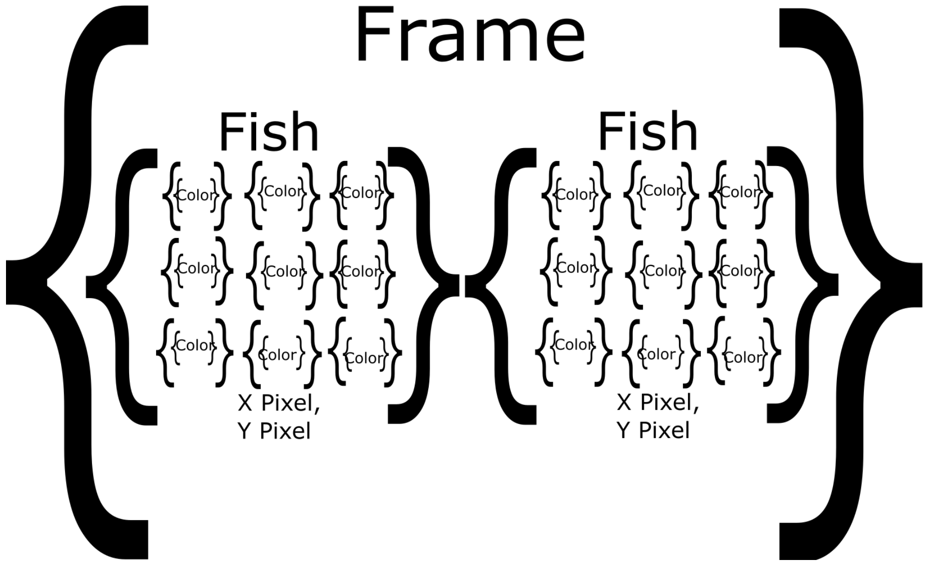


Figure 9: The shape of the array that holds the data

D Histogram Creation

```python
for i in tnrange(60,desc='nonOverlappingRange'):
    for k in range(2):
        countSum=0
        countDif=0
        pairData=[]
        for j in range(*nonOverlappingRange[i]):
            fishPixels = fishU[j][k]
            m,l=np.triu_indices(fishPixels.shape[0],k=1)
            d=np.sqrt((fishPixels[l,0]-fishPixels[m,0])**2+(fishPixels[l,1]-fishPixels[m,1])**2)
            bSum=fishPixels[l,2]+fishPixels[m,2]
            bDif=fishPixels[l,2]-fishPixels[m,2]

            heightValuesSum,_,_=np.histogram2d(d,bSum,bins=(binsDist,binsSum))
            histSum+=heightValuesSum
            countSum+=1
            heightValuesDif,_,_=np.histogram2d(d,bDif,bins=(binsDist,binsDif))
            histDif+=heightValuesDif
            countDif+=1
        histSum/=countSum
        histSumList[i,k]=histSum.copy()
        histDif/=countDif
        histDifList[i,k]=histDif.copy()
```