**Comparing the Effectiveness of Heuristic Algorithms and Support Vector Machine Learning Algorithms to Improve Upon a Haar Classifier**

Ingrid Rumbaugh, [rumbaugi@lafayette.edu](mailto:rumbaugi@lafayette.edu)

Under the Advisement of Dr. Alex Brown, [brownaa@lafayette.edu](mailto:brownaa@lafayette.edu)

GitHub Repository: https://github.com/ingridrumbaugh/ComputerVision

*Department of Mechanical Engineering*

*Lafayette College, Easton, Pennsylvania 18042*

**Abstract Here.**

1. **Introduction**

The original intent of this paper was to compare a heuristic algorithm to a more complex machine learning algorithm utilizing Haar classifiers, in order to identify and track Archerfish. This research is part of a larger project intended to investigate whether or not a robot designed to mimic an Archerfish can train other Archerfish to shoot at a target. Simplifying the process, and therefore decreasing the computing time, for identifying Haar-like features is valuable for the data collection and validation of this larger research project. This paper will highlight the path that this research project followed, in addition to discussing how the chosen algorithms compare to previous work.

1. **The Process**

The purpose of this section is to provide a roadmap of the topics investigated in this paper. Further sections will go into more detail regarding background, specific coding methods, and results. The figure below illustrates the investigative process taken throughout the Spring 2018 semester. Since the Archerfish are distinct enough against the background (the water in the tank), it was thought that using histogram comparisons would be an effective way to find the Archerfish. First, a combination of edge detection and background subtraction were used before implementing histograms. The intent of this first step was to eliminate any object that isn’t moving – effectively making the Archerfish contrast with a black background. Then, histograms of a single video frame were compared to a “ground truth” histogram taken of an Archerfish. This method is called histogram intersection and will be described in the next section.

Moving on to the third step in Figure 1, a transition was made from simple histograms to histograms of oriented gradients (HOG). This was the next logical step, given that HOG is frequently used in image processing to find edges of irregular shapes, such as the Archerfish that are trying to be identified. In the implementation used for this project, a HOG was found using the Sobel Operator, which seemed to be a better fit for the images being manipulated.

Here, HOG is used as a stepping stone into Support Vector Machines (SVMs). This is a type of linear or non-linear machine learning algorithm that can be used to differentiate objects. Although the intent of this research was to investigate replacing such machine learning algorithms, using one would enable this project to differentiate Archerfish not only from the background of the tank, but other fish as well.

|  |
| --- |
|  |

**Figure 1.** Progression of Algorithms Investigated.

1. **Background: Heuristic Algorithms**

In the interest of thoroughness, the background for the basis of this research will be explored – beginning with Haar-like features. Haar-like features are simply digital image features used in object recognition. A Haar feature considers “adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region, and calculates the difference between these sums.” [1]. While there are many types of Haar-like feature descriptors, an example of one can be seen in Figure 2.

|  |
| --- |
|  |

**Figure 2.** Example of a Haar-like feature descriptor [2].

However, a Haar classifier can take a lot of time to train and can quickly become a very complex process. Improving upon this method by finding a simpler way to identify Archerfish is very valuable.

As shown in Figure 1, the first method that was tested was background subtraction and edge detection. Python’s OpenCV2 was one of the main libraries used in this project due to it’s versatility. The *BackgroundSubtractorMOG2* method was used to help eliminate all objects in the frame that were not moving. The *BackgroundSubtractorMOG2* method is a “Gaussian Mixture-based background segmentation algorithm” [3]. An important feature of this algorithm is that it selects the appropriate number of gaussian distribution for each pixel. This provides better adaptability to varying scenes due to features like illumination changes [3]. ­

Once the background is removed, histogram intersection is implemented. This was chosen because the Archerfish do not blend into the background. The intersection algorithm was used from the GitHub repository in Reference 4. The intersection value found is the number of pixels from the model that have corresponding pixels of the same colors in the input image [4]. This intersection is then normalized between 0 and 1 by dividing it by the number of pixels in the histogram. This means that in the results section, a “perfect match” histogram would have an intersection value = 1. A visual representation of intersecting histograms can be seen in Figure 3 [4].

|  |
| --- |
|  |

**Figure 3.** Example of Histogram Intersection [4].

1. **Background: Histogram of Oriented Gradients (HOG) & Support Vector Machines (SVM)**

A Histogram of Oriented Gradient is a feature descriptor that is popular in object detection algorithms. The following steps illustrate how to compute a Histogram of Oriented Gradients (HOG) [5].

1. Global image normalization (Optional)
2. Compute the gradient image in x and y
3. Compute gradient histograms
4. Normalize across blocks
5. Flatten into a feature vector

The first stage applies an image normalization equalization that reduces the influence of illumination effects. However, this step was not used in this instance. The second stage computes first order image gradients. These capture features such as contours, silhouettes, and textures [5]. This means calculating horizontal and vertical gradients, which is easily achieved by filtering the image with the kernels shown in Figure 4.

|  |
| --- |
| https://www.learnopencv.com/wp-content/uploads/2016/11/gradient-kernels.jpg |

**Figure 4.** Kernels Used to Calculate HOG.

Since one of the main libraries used in this project is OpenCV, this second step can be achieved by using the Sobel operator with a kernel size of 3. The Sobel operator is a discrete differentiation operator that computes an approximation of the gradient of an image intensity. It uses joint Gaussian smoothing so it is more resistant to noise [7]. Figure 5 shows how the Sobel operator was implemented in this method.

|  |
| --- |
|  |

**Figure 5.** Code Used to Implement Sobel Operator to Create a HOG.

Next, the magnitude and direction of the gradient is calculated, which was done using OpenCV’s *cartToPolar* method. The following equations illustrate how the magnitudes and directions are found for computing the HOG [6].

|  |
| --- |
|  |

**Figure 6.** Equations for Calculating Magnitude & Direction from Sobel Operator.

In stage three, the image is divided into small regions called “cells”. For each of these cells, a 1-D histogram of edge orientations is created. The combined histogram forms the basic “orientation histogram” representation. This step produces a sort of encoding that is “sensitive to local image content while remaining resistant to small changes in pose or appearance [6].” Next, the data needs to be normalized. This step makes the algorithm less sensitive to illumination and shadowing. Normalization is performed by “accumulating a measure of local histogram ‘energy’ over local groups of cells that are called ‘blocks’ [6].” This result is used to normalize each cell. The final step in creating a Histogram of Oriented Gradients is to create a combined feature vector using Python’s OpenCV library.

In order to implement HOGs, a more complicated binary classifier is required. So far, a linear Support Vector Machine was chosen. The figure below shows the flow from implementing HOG to SVM.

|  |
| --- |
|  |

**Figure 7.** Flow Path from HOG to Linear SVM [8].

A Support Vector Machine (SVM) is a “supervised machine learning algorithm that is mostly used in classification problems” [9]. In this algorithm, each data item is plotted (these are the data from the HOG) and perform classification by finding the hyperplane that best differentiates the two classes of data. SVMs maximize the margin around the separating hyperplane. This decision is specified by a subset of training examples – in this case many images of Archerfish and many images of other objects. The figure below illustrates the intent of maximizing the margin in an SVM [10].

|  |
| --- |
|  |

**Figure 8.** Illustration of Margin Maximization for a Linear SVM.

A linear SVM was chosen as an alternate method to improve upon a Haar classifier because SVMs use a subset of training points in the decision function (these are the support vectors), so it is memory efficient. SVMs are also effective in cases where the number of dimensions is greater than the number of samples. This means that SVMs are versatile, in addition to being accessible by libraries such as *SciKit-Learn.*

1. **Histogram Intersection Methods**

Throughout all of the methods for testing this heuristic algorithm, a “ground truth” histogram was created using an image of an Archerfish taken from the current video frame. In later methods in an effort to increase reliability, three separate ground truth histograms were computed from three separate Archerfish. One of the ground truth histograms can be seen below in Figure 9.

|  |
| --- |
|  |

**Figure 9.** Ground Truth Histogram of Archerfish.

Once a histogram is taken of a section in the frame, it is overlapped with the ground truth histogram. A snippet of code in Figure 10 shows the histogram intersection method used in this algorithm. Again, this method was inspired by code from the repository in reference 4.

|  |
| --- |
| **Figure 10.** Histogram Intersection Method. |

1. The first approach to implementing this heuristic algorithm was to sweep a rectangle of size 40 x 40 pixels, iterated in steps of 5 over the frame. To test the functionality of this program with video, a webcam was used. This program was then implemented with one frame from a video of Archerfish in a tank. However, problems arose when running this program with the bag files of Archerfish swimming. The movement of the fish was not enough to yield significant results from intersecting histograms.
2. The second approach was to repeat the process from before, but without removing the background of the frame. As stated previously, a perfect match would yield an intersection value of 1. However, since this is unrealistic to expect this, intersection values were tested from 0.5 through 0.8, iterating in steps of 0.05.
3. The final approach was to use three ground truth histograms with the intent of allowing a higher rate of ‘matches’ to fish. The hope was that this would enable the algorithm to identify fish in multiple orientations, instead of simply a fish facing towards the right in a perfectly identifiable pose. This method required that the test histogram matched at least two of the three histograms, with an intersection value that was iterated the same as in method 2.
4. **Heuristic Algorithm Results**
5. Background Subtraction + 1 Ground Truth Histogram

This method did not yield significant results when implemented with the Archerfish tank videos. In addition, when this program was run through ROS (Robot Operating System), it was unexpectedly slow. Even though this method was never able to be fully tested, some intermediate results are shown below to illustrate the overall concept. These figures are from webcam testing and show the pre-processed, threshold frame, the final frame with the background subtracted, and the active histogram of the frame. These can be seen in Figures 11, 12, and 13.

|  |
| --- |
|  |

**Figure 11.** Threshold Frame.

|  |
| --- |
|  |

**Figure 12.** Final Frame with Background Subtraction.

|  |
| --- |
|  |

**Figure 13.** Active Histogram of Webcam Frame.

1. 1 Ground Truth Histogram without Background Subtraction

This method was tested against a single video frame with increasing levels of histogram intersection. Figure 14 shows the fish (taken directly from the frame that this algorithm was tested on) that was used for the ground truth histogram.

|  |
| --- |
|  |

**Figure 14.** Fish Used for Ground Truth Histogram.

A selection of frame screenshots are shown below. This selection shows how the algorithm performed with an increasing value of histogram intersection. When the algorithm has identified something as an Archerfish, it prints a green rectangle over the block of pixels. Ideally, the algorithm would find what it thinks is multiple fish and print multiple rectangles over a single fish. This could eventually be mitigated by replacing these overlapping rectangles with one rectangle, that would ideally be an Archerfish.

|  |
| --- |
|  |

**Figure 15.** Algorithm Run on Frame with Required Intersection = 0.6.

|  |
| --- |
|  |

**Figure 16.** Algorithm Run on Frame with Required Intersection = 0.7.

|  |
| --- |
|  |

**Figure 17.** Algorithm Run on Frame with Required Intersection = 0.8.

|  |
| --- |
|  |

**Figure 18.** Algorithm Run on Frame with Required Intersection = 0.85.

It can be seen that as the required histogram intersection value increases, the amount of “fish” the algorithm finds decreases. After the required intersection value was increased to 0.9, there were no matches found. The promising aspect of this method is that at a threshold of 0.85, the algorithm finds only fish. However, it does not identify all of them. Surprisingly, even at an intersection value of 0.6, the algorithm does not identify a fish in the lower left corner that would appear to be a perfect match to the ground truth histogram. This could have to do with the orientation of the fish, or the fact that the RGB ground truth histogram for an Archerfish (see Figure 5) does not have enough unique features. The only unique aspect of this histogram is a single spike of red, green, and blue channels. However, this does not appear to be unique enough for the algorithm to properly separate fish from the background of the tank.

One approach to this problem was to take multiple ground truth histograms of multiple Archerfish in different orientations. This will hopefully mitigate errors due to fish orientation and provide a more unique set of data to compare to. These results will be discussed in the following section.

1. 3 Ground Truth Histograms

This final heuristic method was again tested against a single video frame with increasing levels of histogram intersection. Figure 19 shows the three fish (taken directly from the frame that this algorithm was tested on) that were used for the ground truth histograms.

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Figure 19.** Archerfish used for Ground Truth Histograms.

A selection of frame screenshots are shown below. This selection shows how this second algorithm performed with an increasing value of histogram intersection. For this algorithm, at least two of the ground truth histograms need to match the rectangle of pixels in question. Ideally, the algorithm would find what it thinks is multiple fish and print multiple rectangles over a single fish. While this seems like a problem, it is a good indication that the algorithm is identifying fish and this cluster of rectangles could be mitigated by replacing them with one rectangle, that would ideally be an Archerfish.

|  |
| --- |
|  |

**Figure 20.** Algorithm Run on Frame with Required Intersection = 0.6.

|  |
| --- |
|  |

**Figure 21.** Algorithm Run on Frame with Required Intersection = 0.7.

|  |
| --- |
|  |

**Figure 22.** Algorithm Run on Frame with Required Intersection = 0.75.

While this algorithm started out with promising results, finding almost all of the Archerfish in the tank, this method failed at a lower required intersection than the algorithm using only one ground truth histogram. In addition, when the algorithm identifies only one “fish”, it can be seen clearly that the rectangle is not in the vicinity of an Archerfish. With these results, it was decided that this purely heuristic algorithm was unable to replace a Haar classifier. Additional reasons for why this algorithm failed will be discussed in Section 9.

As shown in Figure 1, the next step in this process to improve upon a Haar classifier was to investigate the effectiveness of histograms of oriented gradients. Instead of using RGB histograms, a histogram taken of the magnitude and direction of each pixel in a frame is used. The following section will discuss the methods used so far in this process.

1. **HOG & SVM Methods**

First, a HOG of both the ‘ground truth’ fish and a single video frame was computed using mostly the OpenCV library in Python. As outlined in Section 4, the process of computing a HOG involved converting the image to grayscale, applying the Sobel operator, and extracting the magnitude and angle from that data.

Then, this data would be used by the Linear SVM, implemented using Python’s SciKitLearn library. First, source parameters need to be chosen, which is not a concept that this report will cover - see references 11 and 12 for more information. Next, images of Archerfish and non-Archerfish were loaded into the SVM. These training images were supplied from Dr. Brown’s GitHub Repository (see Reference 13). The algorithm then extracts features of both fish and non-fish. Finally, an instance of a LinearSVC is created and the training variables created from the images are inserted.

1. **HOG & SVM Results**

Training the SVM has not yet been completed; however, a source has been provided that shows the complete process and how it performed with vehicle classification. (See references 11 and 12 for more discussion of this technique). Although the Linear SVM has not been trained yet, HOG data has been computed and can be seen in the figures below.

|  |
| --- |
|  |

**Figure 23.** HOG Plot of Single Archerfish.

|  |
| --- |
|  |

**Figure 24.** HOG Plot of Non-Archerfish.

|  |  |
| --- | --- |
|  |  |

**Figure 25.** HOG Plot of Frame (Left) with Combined Sobel Operator (Right).

It can be seen from Figures 23 and 24 that the HOG of a single Archerfish is quite different from the HOG of a non-Archerfish. The HOG in Figure 24 is taken from a section of pixels on the right side of the tank. Figure 25 shows a HOG plot of the whole frame, along with the combined Sobel operator. These plots are promising, because it will allow the SVM to easily distinguish the fish from the rest of the frame. By inspection, a linear SVM may work well because the features in these HOGs are linearly separable.

1. **Discussion & Conclusions**

In the pursuit of trying to improve upon and simplify a Haar classifier, multiple interesting object identification methods were explored. While the simplicity of histogram intersection was appealing, the data did not yield significant enough results to act as a legitimate replacement of a Haar classifier. However, perhaps there are other similar methods involving histogram intersection that can be explored, such as providing different or more ground truth histograms. The major fault with this method is that the Archerfish do not have distinct enough features to make an RGB histogram feasible.

There is, however, great promise in using Histograms of Oriented Gradients in conjunction with a linear Support Vector Machine. The preliminary data is a good start, and future research will be able to determine whether a linear SVM can be trained to identify and track Archerfish. However, if a linear SVM does not significantly improve upon the current Haar classifier, a non-linear SVM can also be investigated.

This project is a great start to the multitude of object identification methods that exist and are made extremely accessible by libraries such as Sci-Kit Learn and OpenCV. While the intent of this paper was not for publication, a lot of great research has been accomplished, and will continue to be improved upon in the upcoming months.

**References**

[1] <https://en.wikipedia.org/wiki/Haar-like_feature>

[2] <http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_haar.html>

[3] <http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_bg_subtraction/py_bg_subtraction.html>

[4] <https://mpatacchiola.github.io/blog/2016/11/12/the-simplest-classifier-histogram-intersection.html>

[5] <http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html>

[6] <https://www.learnopencv.com/histogram-of-oriented-gradients/>

[7] <https://docs.opencv.org/3.1.0/d2/d2c/tutorial_sobel_derivatives.html>

[8] <http://web.eecs.umich.edu/~silvio/teaching/EECS598_2010/slides/09_28_Grace.pdf>

[9] <https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/>

[10] <http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf>

[11] <https://medium.com/@mithi/vehicles-tracking-with-hog-and-linear-svm-c9f27eaf521a>

[12] <https://github.com/mithi/vehicle-tracking-2/blob/master/ipynb/feature_sourcer_test.ipynb>

[13] <https://github.com/Alexanderallenbrown/fishtracker_ros>