Real-World Aim Assist

object detection using Cascade Classifier

Annafabia Gai | Cristina Lakasz | Simone Russolillo | Filippo Wang
Sapienza University of Rome

ABSTRACT

This project aims to build a prototype of a real-world gun aim assist.

Using a toy gun, we were able to build an aim assistant to guide marksmen, through an auditory signal, in identifying and aiming at our targets, soda cans. Thus, offering the user enhanced targeting capabilities and improved operational efficiency. The objective of this paper is to present the design, development, and evaluation of the prototype we created.

1 INTRODUCTION

Whether it be in competitive shooting competitions, personal defense situations, or military operations, the ability to consistently hit targets with precision and efficiency is a coveted skill sought by marksmen of all levels. Recognizing the importance of enhancing shooting accuracy and reducing target acquisition time, our project endeavors to develop a gun aim assistant.

The motivation behind this project arises from the desire to provide shooters with a practical and intuitive tool that can increase targeting capabilities and improve operational efficiency by maintaining a precise and accurate aim. The prototype we developed incorporates a camera-based sensor system, image processing algorithms, and a user interface for real-time feedback and adjustments. The advancement of artificial intelligence offers powerful tools to augment human potential, we envision a future where shooters, both in professional and recreational contexts, can enhance their skills by optimizing their performance and aim.

2 BACKGROUND

A considerable part of the project consisted in training an object detection model. Object detection is the ability to locate an instance of an object in an image or footage. We decided to adopt the machine learning-based approach of Haar feature-based cascade classifiers provided by OpenCV. Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" published in 2001. [19]

Haar Cascade Classifier. Haar features [Fig.1] are simple rectangular patterns used to describe the visual characteristics of objects and are derived from a training set of positive and negative images. Positive images are the ones actually containing instances of the object we are interested in, whereas the negative ones are arbitrary images not containing it. Haar features, which are like a convolutional kernel, are extracted from the training images by computing the correlation between the image and the Haar feature filter. This is determined by the difference between the sum of all the image pixels lying in black region and the sum of all the image pixels lying in the white region of the Haar feature. A preliminary step to do so is filtering the images in grayscale, so that each pixel has a value between 0(white) and 1(black) [7]

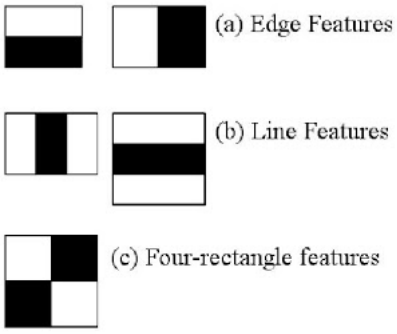


Figure 1: Haar Features

Image classifiers work by using a sliding window across the image, a rectangular region of fixed width and height, that determines the smallest detectable object. The classifiers perform calculations on the pixels contained in the sliding region, so the bigger the window size, the greater the computational complexity. Integral images are used to ease computations, here the sum of pixel intensities within any rectangular region can be computed efficiently using only four values, regardless of the region size, by summing the value of the left top corner of the rectangle with the right bottom one and then subtracting the right top and the left bottom ones. This is true because a given pixel in the integral image is the sum of all the pixels on the left and above it. [18] Adaboost algorithm is also used as a means to further improve computational efficiency. Adaboost is a statistical classification meta-algorithm that can be used together with other learning algorithms to improve performance. It applies the features to the images separately in order to create Weak Learners to select a subset of informative features and assign weights to each. The training images are then processed sequentially with Haar feature cascade, a sequence of classifiers, where each classifier consists of a subset of the informative features. [20] [18] Attentional cascade further improves performance by moving to the subsequent windows, of an image, every time the window does not pass the classifier's threshold. Haar features cascade then achieves accurate and real-time object detection by efficiently processing regions of interest. [16]

Training a Cascade Classifier. To train a Cascade Classifier, we first need to collect the dataset of positive and negative images. The ratio between positive and negative samples is a key factor in the training process, and the different values can work best in different situations. In order to train a Cascade Classifier, we require a dataset

AI Lab project, Sapienza University of Rome,
2023.

of training images and two text files indicating their location. Negative samples' text file only contains the path of each image. Positive samples need a similar description file, which not only contains the path of each image but also the number of appearing objects that we want to detect and the coordinates of bounding rectangles of each object. OpenCV provides a useful tool to ease this process. The OpenCV annotation program [11], in fact, outputs the text file for the positive images displaying every image present in a folder and allowing the user to draw rectangles on each of the objects we want to detect: *opencv_annotation –annotations=/path/to/annotations/file.txt –images=/path/to/image/folder/*. The next step is to create a vector file from the positive data file using the *opencv_createsamples* [12] utility: *opencv_createsamples.exe -info pox.txt -w 24 -h 65 - num 1000 -vec pos.vec* where -w and -h indicate the detection window size, num indicates the number of vectors to create, which should be a number larger than the amount of rectangles drawn. To train the model, we simply run: *opencv_traincascade.exe -data folder/ -vec pos.vec -bg neg.txt -w 24 -h 65 -numPos 200 -numNeg 100 -numStages 10.* [13] Where -vec indicates the location of the vector file of positive images, -bg (the background) the location of the negative images, -w and -h the size of the window size. -NumPos and -numNeg indicate the number of positive and negative samples to be used for the training, -numStages the number of stages to train, the more the stages the longer the training will take and the better the model will perform. For each stage of the training, a table is printed on the terminal showing the weak layer number (N), the hit rate (HR) and the false alarm (FA), thus showing the performance of the model in each layer. The result is a .XML file of the trained model. [14] [3] Other useful parameters of the training function are *-maxFalseAlarmRate <max_false_alarm_rate>*, specifying the minimal desired hit rate for each stage of the classifier, and *-maxFalseAlarmRate <max_false_alarm_rate>*, specifying the maximal desired false alarm rate. The values of these parameters strongly depend on each specific application and on the characteristics of the dataset. It is important to carefully tune these parameters through experimentation and validation, so to achieve the desired trade-off between accuracy, efficiency, and robustness of the model.

ESP32-CAM. The second step in the creation of the project was programming a camera in order to access and manipulate its footage. We used a ESP32-cam [2], a microcontroller from Espressif, consisting of a compact board with ribbon cable Camera Serial Interface with a small camera, a microSD card slot and Bluetooth and Wi-Fi connectivity. The Arduino IDE is an open-source software platform that provides an interface for writing, compiling, and uploading code to Arduino-compatible boards, including the ESP32-CAM. [Fig.2]

3 DESIGN

By using soda cans as our targets, we could build an object detection model, employing Haar Cascade. An ESP32-CAM, attached to our toy gun, would let us manipulate the live stream from the gun's point of view and subsequently give shape to our project. Lastly, by running tests on our prototype and collecting the results we could evaluate the performance of it.

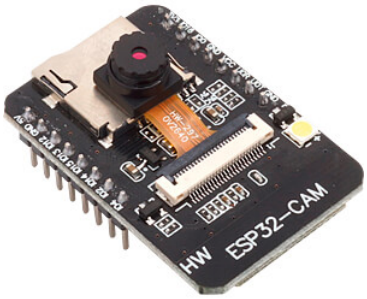


Figure 2: ESP32-CAM

- *Create an object detection model.* We first would need to collect data by taking numerous pictures of several environments, both containing and not containing soda cans. Using Haar Cascade we could then train our model on top of the dataset constructed by us.
- *Interact with the video footage.* By setting up the ESP32-CAM and by connecting to the local IP address where the footage was streamed on, we could be able to interact with the live video footage from the Web Server. The live stream could be used as the input of our object detection model and an auditory signal could indicate whether to move the gun *up, down, left* or *right* in order to have the target on the calibrated crosshair and finally give the *shoot* signal to advise the marksman to pull the trigger.
- *Assemble the components.* Applying the ESP32-CAM on a toy gun and using the live video stream as the input of our model, we could be able to output the auditory signal in order to help the sniper detect where the target was located.
- *Test.* We could then be able to understand how well our prototype would work by testing it in different environments and evaluating the results.

4 IMPLEMENTATION

4.1 Create an object detection model

Training an object detection model that would accurately identify soda cans was a laborious process. We faced numerous challenges with both underfitting and overfitting models, but through persistent refinements, we were able to successfully improve the performance of our model. Here we will explain the difficulties we faced and how we overcame them.

The first models. Our first objective was to create our own dataset, collecting *884* images, (*644* the positive, *240* the negative). We used 4 different types of 33cl soda cans: Pepsi, CocaCola, Fanta, Sprite. We took pictures of them with the logo facing the camera, in several environments and lighting. Some pictures contained only one can, and others contained two or three, placed in different positions. We annotated each image by drawing bounding boxes on each soda cans to save the coordinated of each can in a tuple *(x, y, w, h)*, and then created the text files of the negative and positive images, and the vector file of the latter. We then trained the model.

At first, we did not apply any filters to the images before the training. We then thought of applying a color thresholding filter, as in this way soda cans' colors features would have been emphasized. However, we soon realized that in Haar Cascade, colors were not a relevant feature in the model, so we removed this filter and applied a grayscale filter on every image. Another filter we applied that we thought might well suit our purpose is Gaussian Blur, to smooth the images so as to reduce the noise level of the background. Other filters we experimented with were, canny [8] (for edge detection) and adaptive thresholding [10] (for image segmentation and noise reduction), but we soon removed these as they were not increasing detection accuracy.

Once we collected and processed our training data, we trained the Cascade Classifier, as explained in the background section. We trained a dozen of different models with this dataset, varying parameters such as the feature detection window size and the degree of the blur. By modifying the size of the sliding window we found that the *24x65* pixel ratio was the one working the best, being in fact the actual shape ratio of the soda cans. By blurring the training images, soda cans would be effectively identified during test time only if they were relatively close to the camera however, as they were moved further away, relevant features of the can would have been smoothed out as well, resulting in the loss of important details and leading to inaccurate detections. Another filter we used was Histogram Equalization, which enhances the contrast in an image by redistributing the pixel intensities to cover the entire dynamic range. [15]

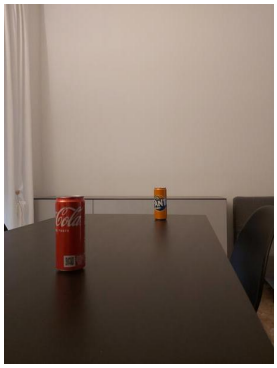
These models were all underfitting. In fact, by testing the models on a live video, rectangles would have been drawn not only on soda cans but also on other irrelevant objects present in the frame. As a way to improve detection accuracy, we decided to try to train a model using a different dataset in order to highlight what a soda can was not.

Data augmentation. To increase the diversity of data available for training models, we used data augmentation on a new set of negative and positive images. [21] We incorporated two public datasets of negative images [5] [4]. Applying a threshold of *300x300* pixels and *700x700* pixels, We removed too small and too large images so to set a uniform range of image sizes in the dataset. We took *6* high resolution pictures of, CocaCola, Sprite and Fanta (we removed Pepsi), for a total of *18* images. Soda cans were positioned on a black desk with a white background, in a well lighted room. We then cropped the images, in order to only have the soda cans, which we resized to have size *24x65* pixels. We then used opencv_createsamples application to not only create the vector file of positive images but to also integrate the dataset with artificially generated positive samples. [Fig.3] These are constructed from a given single object image by randomly rotating it around all three axes, changing the image intensity, as well as placing the image on arbitrary backgrounds, provided by the negative images. [14] Thus, having a dataset of around *6000* positive samples and around *3000* negatives, we trained different models, varying the filters and the parameters applied. We soon recognized that using only artificially generated images lead to an overfitting model. In fact, during the training phase, the false alarm rate dropped almost immediately to *0* as the model was quickly able to detect the object, because each of the 6 images of soda cans was perfectly cropped, and artificially generated. In the testing phase, these models would successfully detect the target only sometimes and persistently identify objects that were not the targets, as if they were.

Figure 3: Synthesized Image

Best performing model. After extensive experimentation, We figured that the best performing model was the 4^{th} one we generated. It was trained using the manually created dataset, suggesting that in this situation, real images lead to a better model compared with artificially generated ones. After testing with different ratios of positive to negative samples, the one that worked best in our case was *2:1*. The filters for the final model only consisted of grayscale and histogram equalization. [Fig.4a, Fig.4b]

(a) GrayScale and HistEq



(b) No filters

4.2 Interact with the video footage

Configuring the ESP32-CAM. In order to capture and retrieve visual data in real time, we used an ESP32-CAM. To effectively use the camera we needed to program it. The first step was connecting the camera to our computer, to do so we used a FDTI adapter and we followed a specific module that indicates which GPIO pins to link. [17] [Fig.5] To interactively control the ESP32-CAM we used the Arduino IDE [1], on which we installed specific Boards for ESP32. These are a collection of files needed to compile and upload sketches for a board. The sketch which was right for us was Sample Sketch, provided by Arduino IDE, under the name of CameraWebServer, this allowed the interaction with the camera by selecting the CAMERA_MODEL_AI_THINKER camera model. To correctly configure the ESP32-CAM we needed to make sure

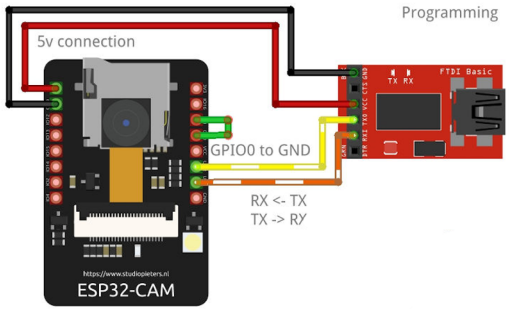


Figure 5: connection module FTDI with ESP32-CAM

the camera and our computer were connected to the same Wi-Fi to successfully transmit the live streaming. The port to be selected was the USB one connecting the FTDI chip. After having uploaded and compiled the sketch, we obtained the IP address of our ESP32-CAM, which would let us access the live streaming. To get the IP address, we opened the Serial Monitor and set it to a baud rate of 115,200 bps, by then pressing the Reset switch, located on the board module, we were able to see the IP of the camera as URL, indicating that the board was connected to the network. By inserting this URL in the search bar of our web browser, we were able to see the webpage of the camera, through which we could have access to our ESP32-CAM settings. Finally, via the Start Stream button we could start the live stream, and using the OpenCV function: Cv2.videocapture [9] we were able to capture the video stream, on top of which we could perform object detection. We soon understood that a small antenna could have been useful in order to amplify the Wi-Fi signal and have better data transmission, so we integrated one into our prototype.

The auditory aim guide. After having detected the target, the program output an auditory signal: *up, down, left, right, shoot* to indicate where to move the camera in order to have the target placed in the center of the video frame. As soon as the target was in place, the program would advise to shoot, with the respective signal.

Figure 6: Prototype

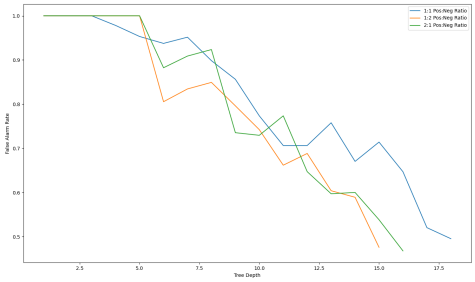
4.3 Assemble the components

Finally, we had to assemble our prototype. We used a toy gun with foam bullets. [6] We attached on it a small portable charger to charge the camera, the ESP32-CAM, and the antenna using black tape. [Fig.6 We then calibrated the camera and started testing the object detection model integrated with our toy gun, targeting our soda cans.

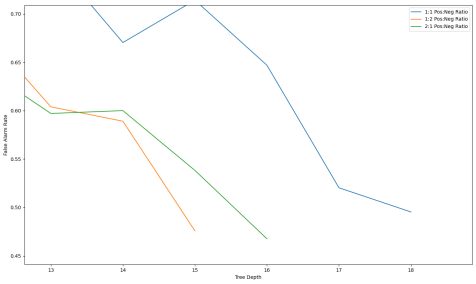
5 EVALUATION

5.1 Best performing model

After training numerous different models, the best performing one was the 4^{th} one we generated. It was trained using the manually created dataset, a *2:1* positive to negative samples ratio, and the filters applied were gray scale and histogram equalization. Following is a line plot comparing different positive to negative ratios' influence on the model's performance. It shows how the 2:1 ratio leads to a smaller false alarm rate, resulting in a more accurate model. [Fig.7a, Fig.7b]



(a) N-FAR comparing pos:neg ratio



(b) Zoom In

5.2 Successfully working prototype

After having assembled our prototype, we could test it ourselves. We could input the gun's point of view in our object detection model, and it would output the desired auditory assistance to efficiently aim at soda cans. We achieved our goal to create a real-world aim assist. [Fig.8]

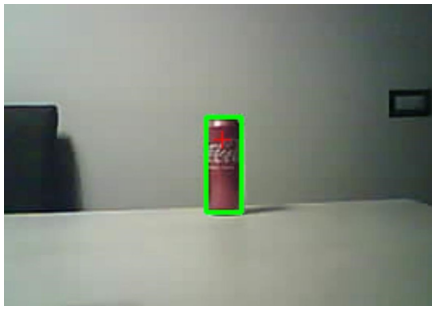


Figure 8: Stream Output

6 CONCLUSION

After considerable experimentation, we were able to create an object detection model to recognize soda cans. After numerous attempts, we managed to select the dataset, filters, and parameters that best worked for us. We configured a camera and made refinements to the hardware implementation of the project, so to obtain a refined prototype. We were able to apply our object detection model on top of a real time stream and provide an auditory feedback as an aiming support.

Extensive testing and evaluation was conducted to assess the effectiveness and reliability of this aim assist. We can then affirm that we have successfully achieved the goal of creating a prototype that could guide marksmen in the detection of the target. Furthermore, the proposed aiming aid prototype could have great potential for application in military and recreational shooting situations, providing users with enhanced aiming capabilities and operational efficiency. Looking ahead, the integration of AI-powered aim assist could ultimately revolutionize all target shooting environments.

REFERENCES

[1] Arduino. [n. d.]. Arduino IDE. https://www.arduino.cc/
[2] Espressif. 2019. ESP32-CAM and Other Cool Projects on RNT. https://www.espressif.com/en/news/ESP32_CAM
[3] Learn Code By Gaming. 2021. Training a Cascade Classifier - OpenCV Object Detection in Games. https://www.youtube.com/watch?v=XrCAvs9AePM
[4] JoakimSoderberg. [n. d.]. haarcascade-negatives. https://github.com/JoakimSoderberg/haarcascade-negatives
[5] Muhammad Khalid. [n. d.]. Negative Images for Haar Cascade. https://www.kaggle.com/datasets/muhammadkhalid/negative-images
[6] Nerf. 2012. Firestrike (N-Strike Elite). https://nerf.fandom.com/wiki/Firestrike_(N-Strike_Elite)
[7] First Principles of Computer Vision. 2021. Haar Features for Face Detection | Face Detection. https://www.youtube.com/watch?v=ZSqg-fZJ9tQ
[8] OpenCV. [n. d.]. Canny Edge Detection. https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html
[9] OpenCV. [n. d.]. cv::VideoCapture Class Reference. https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html
[10] OpenCV. [n. d.]. Image Thresholding. https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html
[11] OpenCV. [n. d.]. opencv_annotation. https://github.com/opencv/opencv/tree/3.4/apps/annotation
[12] OpenCV. [n. d.]. opencv_createsamples. https://github.com/opencv/opencv/tree/3.4/apps/createsamples
[13] OpenCV. [n. d.]. opencv_traincascade. https://github.com/opencv/opencv/tree/3.4/apps/traincascade
[14] OperCV. [n. d.]. Cascade Classifier Training. https://docs.opencv.org/3.4/dc/d88/tutorial_traincascade.html
[15] OperCV. [n. d.]. Histogram Equalization. https://docs.opencv.org/3.4/d4/d1b/tutorial_histogram_equalization.html
[16] OperCV. [n. d.]. OpenCV: Cascade Classifier. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
[17] Random Nerd Tutorial. [n. d.]. How to Program / Upload Code to ESP32-CAM AI-Thinker (Arduino IDE). https://randomnerdtutorials.com/program-upload-code-esp32-cam/
[18] Mrinal Tyagi. 2021. Viola Jones Algorithm and Haar Cascade Classifier. https://towardsdatascience.com/viola-jones-algorithm-and-haar-cascade-classifier-ee3bfb19f7d8
[19] Paul Viola and Michael Jones. 2001. Rapid Object Detection using a Boosted Cascade of Simple Features. (2001). https://ieeexplore.ieee.org/abstract/document/990517/authors#authors
[20] Wikipedia. [n. d.]. AdaBoost. https://en.wikipedia.org/wiki/AdaBoost
[21] Mayank Yogi. 2020. Data Augmentation Techniques using OpenCV. https://medium.com/analytics-vidhya/data-augmentation-techniques-using-opencv-657bcb9cc30b