Independent Study: Drone Detection

Wyatt Hough

Advisor: Dr. Kuan-Lun Chu

**Introduction**

The goal of this project was to detect and track commercial drones flying in an open outdoor environment. This was accomplished by using the computer vision library OpenCV to process images and train a Haar Cascade classifier to recognize these drones. The classifier works by superimposing positive images over many different negative images and detecting features specific to these training samples. Different weights are applied to misclassified images as the training stages increase until the required accuracy is achieved.

**Procedure**

The process for training is detailed below. All commands were run in the Windows Command Prompt application unless specified otherwise. We made three iterations of the training model using different sets of images for each run.

Step 1: Obtain positive and negative images of the drone. The positive images were gathered by taking pictures of the drone and removing the background of the image leaving the drone with a uniform white background. The image was then turned to grayscale and shrunken to a 48x48 pixel resolution. The small image size was due to the training algorithm being extremely memory intensive and there was a limited amount of RAM on the computer we used. Figure 1 shows an example of positive images blown up for easier viewing. In the final training run, there were 30 of these positive images and 830 negative images. The negatives are simply pictures of anything without drones in them. Most of them were similar to the image in Figure 2.



Figure 1. Examples of positive images



Figure 2. Example of negative image

Step 2: Create samples from each positive image. This is where the positive image is superimposed onto the negatives to create training samples. The result was still a 48x48 resolution image of the drone, but it is overlaid onto a random section of one of the negative images. The program keeps track of which image and section this sample is taken on so it can compare the exact same window with and without the drone in it. An enlarged sample is show in Figure 3. Given that there were 30 positive images in run 3, we produced 70 samples per image for a total of 2100 samples. The command had to be run 30 times for each of the positive images so we created a batch file which simply ran all 30 commands in a row. The exact command used is shown below along with descriptions for each of the parameters.



Figure 3. Example of positive sample

“opencv\_createsamples -img C:\opencv\DroneTracking\Run3\Positives\pos(1).png -vec C:\opencv\DroneTracking\Run3\vec\samples1.vec -bg C:\opencv\DroneTracking\Run3\negatives.txt -num 70 -bgcolor 255 -bgthresh 20 -maxidev 40 -maxxangle 0 -maxyangle 0 -maxzangle 2 -w 48 -h 48 -rngseed 101”

-img (file path to the positive image)

-vec (file path for the output .vec file)

-bg (path to a text file containing all of the negative image’s filenames)

-num (the number of samples generated for the given image)

-bgcolor (the RGB value of pixels in the positive image which are made transparent when overalaid on the negative, the value is 255 for white)

-bgthresh (the +/- range of the bgcolor value to be made transparent, a value of 20 would mean any pixels between 235 – 275 are made transparent)

-maxidev (the maximum deviation in intensity of the positive image, this makes the image brighter or dimmer randomly for each sample)

-maxxangle (the maximum rotation angle on the x axis in radians, kept at 0)

-maxyangle (the maximum rotation angle on the y axis in radians, kept at 0)

-maxzangle (the maximum rotation angle on the z axis, value of 2 rotates the drone clockwise or counterclockwise for more variety in the samples)

-w (the width of the image in pixels)

-h (the height of the image in pixels)

-rngseed (the seed for the random number generator, changed this value for each image)

Step 3: Merge the resulting .vec files. Next, we simply needed to combine all 30 of the files into one big .vec file to be fed into the final training command. To do this, we used a python script from the GitHub repository titled *mergevec* by user “wulfebw”. The exact command is pasted below. It takes in the path of the folder containing all the individual files and outputs a single .vec file containing all of the image data.

“C:\opencv\mergevec.py -v C:\opencv\DroneTracking\Run3\vec -o merged.vec”

Step 4: Train the cascade. The final step was to train the Haar Cascade classifier. One important note was to input a smaller number of positive and negative samples than what we had made. This was because the program ends up consuming more than the stored values in order to meet the specified minimum hit rate. Below is the exact command run to train our cascade along with the given parameter definitions. The 1600 and 800 values for the numbers of positive and negative samples are well below the 2100 and 830 respective values we actually had for our third run. The first two runs used 3000 positives and 1500 negatives. Another factor worth mentioning is that it was recommended to have a 2:1 ratio of positives to negatives when training for best results. Many of the other parameters for this command are left blank meaning they are given their default values. This includes the default minimum hit rate of 0.995 and the maximum false alarm rate of 0.5.

“opencv\_traincascade -data C:\opencv\DroneTracking\Run3\xml -vec C:\opencv\DroneTracking\Run3\vec\merged.vec -bg C:\opencv\DroneTracking\Run3\negatives.txt -numPos 1600 -numNeg 800 -precalcValBufSize 4096 -precalcIdxBufSize 4096 -w 48 -h 48 -mode ALL”

-data <cascade\_dir\_name>

-vec <vec\_file\_name>

-bg <background\_file\_name>

[-numPos <number\_of\_positive\_samples = 2000>]

[-numNeg <number\_of\_negative\_samples = 1000>]

[-numStages <number\_of\_stages = 20>]

[-precalcValBufSize <precalculated\_vals\_buffer\_size\_in\_Mb = 1024>]

[-precalcIdxBufSize <precalculated\_idxs\_buffer\_size\_in\_Mb = 1024>]

[-baseFormatSave]

[-numThreads <max\_number\_of\_threads = 5>]

[-acceptanceRatioBreakValue <value> = -1>]

--cascadeParams--

[-stageType <BOOST(default)>]

[-featureType <{HAAR(default), LBP, HOG}>]

[-w <sampleWidth = 24>]

[-h <sampleHeight = 24>]

--boostParams--

[-bt <{DAB, RAB, LB, GAB(default)}>]

[-minHitRate <min\_hit\_rate> = 0.995>]

[-maxFalseAlarmRate <max\_false\_alarm\_rate = 0.5>]

[-weightTrimRate <weight\_trim\_rate = 0.95>]

[-maxDepth <max\_depth\_of\_weak\_tree = 1>]

[-maxWeakCount <max\_weak\_tree\_count = 100>]

--haarFeatureParams--

[-mode <BASIC(default) | CORE | ALL

--lbpFeatureParams--

--HOGFeatureParams--

**Results**

We ended up making three different runs of the training process using different sets of training images each time. The first run through was trained with images taken of the drone while flying and doing very little image processing before inputting them into the create\_samples command in step 2. The second run used images of the drone taken indoors on a desk and cropped out of the picture for a cleaner look. These were augmented with some images of other types of drones found online for a wider variety of images. And lastly, the third run used images of the drone taken flying outside but with a good bit of preprocessing before being inputted to the create\_samples command. The background was removed, and the image was scaled down beforehand to the 48x48 resolution whereas previously the create\_samples command would do that automatically. Figure 4 shows the differences of the positive images in the three iterations.



Run 1

Run 2

Run 3

Figure 4. Examples of positive images for each run

Since run 3 was focused on using more realistic data, the propellers were cropped out of the positive images as well since we figured the frame rate of the video would not be able to consistently capture propeller motion due to their rapid rotation speed. We also removed all of the negative images which were not pictures of an outdoor environment so that the training samples would have the most realistic backgrounds. This is why run 3 used a much smaller data set. In order to test the three different training models, the same video was fed into the detection program running with a different classifier each time. Runs 1 and 3 produced usable results while run 2 was littered with false alarms. For this reason, we disregarded run 2 and ceased to perform further tests on it. Counting the number of frames that resulted in a false alarm and dividing it by the total number of frames found that run 1 produced an error rate of about 23% while run 3 produced an error rate of 56%. Figure 5 depicts the differences between the two when run on a single image. We expect these rates could be lowered if certain parameters in the detection program were modified. However, run 1 still performed the best by a substantial margin. We would have preferred to run a few more training iterations and change the feature type parameter from Haar to LBP which stands for local binary patterns and is known to be much faster than Haar but a little less accurate. This is because LBP uses integers in its calculations as opposed to floating point decimals.



Run 1



Run 3

Figure 5. Example of run 1 compared with run 3 on the same image

**Conclusion**

In the end, the first run ended up with the most accurate tracking during evaluation. We believe this is because the training data was the most realistic and most accurately represented a drone in a real-world scenario. The training images from run 2 were too artificial and a drone flying outdoors would never appear this clearly. That is why run 2 performed the worst during our tests which only factored in videos and images from real world situations. Lastly, run 3 seemed to have a solid training set but having fewer training samples may have contributed to its shortcomings. The thinking here was that quality images over quantity would lead to a more accurate model. Another important factor is the omission of propellers in the positive images for run 3. As we can see in Figure 5, the propeller areas were where the most false alarms occurred. In the future, it would be beneficial to change the boost parameters such as minimum hit rate and maximum false alarm rate during the training stage and see how these would have affected the classifiers. However, since each training command took up to a week to finish running, it was unfeasible to attempt these different scenarios. As an achievement, the program was run on a laptop and worked well enough to detect drones in a controlled but live setting outdoors.

**Resources**

1. Tutorial on how to use the OpenCV library for object detection <https://docs.opencv.org/3.4.3/dc/d88/tutorial_traincascade.html>
2. Online photo editor used to remove the background of positive images <https://www.photopea.com/>
3. Extra information and tips on how to get best results when training cascade classifier <https://stackoverflow.com/questions/16058080/how-to-train-cascade-properly>
4. GitHub repository of program to merge .vec files

<https://github.com/wulfebw/mergevec>