600100 Computer Vision /
601097 Machine Vision and Sensor Fusion

Identifying road signs

Student ID: 202011724
Due date: **23/03/2023 14:00**

Image Processing Pipeline

My image processing pipeline starts by loading an image/frame from video file. It also loads in 3 variables, Database R,B and W which contain pre-generated signatures used to detect shapes. It then applies a weak gaussian blur. This helps eliminate sharp, rugged edges and some video bitrate artifacts. The image is then split from RGB to HSV. To improve detection of color in varying environments a value and saturation histogram equalization is performed, to boost contrast. This should allow the program to easily work with a low saturation, dark image and a high contrast, high saturation image without any changes. Then I create Red Blue and White layers by binarizing based on hue, saturation and value that was acquired by trial. Initially I then eroded the layers, but after testing I found out that it reduces the accuracy of the result. From each layer I remove values smaller than a certain number of pixels to eliminate noise. Then from the blue and red layers I generate a bounding box for each object remaining. The bounding box is slightly expanded. This is to catch the edges of the image, that could otherwise be cropped.
The next step is detecting red/blue objects. The program splits into two loops. The first one will detect red signs, and the second one blue signs. They both follow the same steps. For each object of color program isolates the are containing the object using the saved bounding box. Then yet again small objects, smaller than a threshold get removed to remove noise, especially in the corners. Then using region props from the edge of the image I generate a list of edges using the build in canny edge detection. Edges below a certain size get removed. I then store each pixel, and centroid of each edge.
As suggested by Andika and Rajif, I using the edge and centroid I calculate and save the angle-distance signature for each edge. The data is then normalized and rescaled, so the x is between -pi and pi and the y is between 0 and 1. The signature is then compared to each signature in my Dataset, which I generated the same way using Dataset_Generator.m and a few simple images. I compare the images by getting the shortest distance from each point to a point in a the pre-generated signatures within the x range. This returns a number (error) which is lower for more probable signature match. This is performed for red and white

objects, and then for blue and white objects. The results are stored in a structure as the name, detected shape and probability of the shape.
I decided to use the angle-distance (R, theta) signature, because it looked like a reliable way of comparing shapes on an image. The alternative would be to compare each pixel. At the time using signatures of the shape seemed like a more robust way of comparing objects. Another reason was because it looked like a good opportunity to learn more about the topic. This was the first time I heard the concept of using a signature of a shape made up of the distance from the center and angle, to estimate a shape, and I wanted to find out more.
At this point all image processing is done and the program needs to pick the most likely sign. For each region deemed a sign it looks at the found signs (red and blue) and checks what their most likely contents are. It then decides what sign it is, storing its bounding box and and name, which can be used to easily display the sign on the image.

Results

The pipeline manages to reliably detect well-lit, well-positioned, easy to read signs. It has no problems with the simple examples. Where is struggles is when the saturation is reversed (sign is dark, background bright), to which I wasn't able to find a solution. It also has trouble detecting smaller images, where the edges overlap. Since I am using the edges to detect shapes, if two edges that should be separate overlap (due to noise/dirtiness of the sign) the pipeline can have issues dealing with it. It works well with images where the saturation or value is not balanced, thanks to equalizing them. The pipeline also works with multiple signs, if they don't overlap. However, when the background of the image is similar to the sign, it struggles to differentiate, resulting in bad results.
Testing the pipeline on videos is difficult, mostly due to the bad performance. Each frame takes a few seconds. The pipeline detects the possibility of an image in most cases, but it's also very good at detecting false signs. This is because the images are badly lit/oversaturated which puts the filters off balance. On top of that making a pipeline that works with image and video quality was harder than expected.
That being said it does manage to detect the signs in some frames of the videos.

Discussion

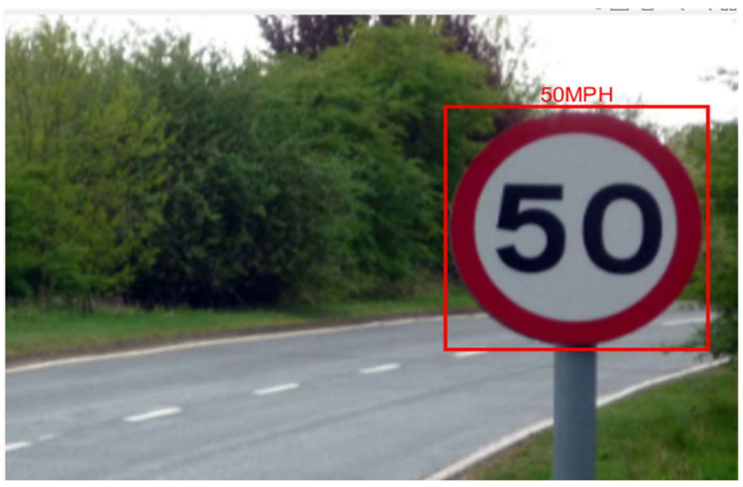
The detection of objects on an image works well. The program detects a lot red/blue shapes that it then tests for signs. The problems it has is specific images, where the saturation of the sing is lower than the saturation of the rest of the image. Detecting the contents of the sign also work great, but only with a few selected signs. For a lot of them the program will label it wrong. It also detects signs, where there isn't any, like in the road, or in windows.
The distance-angle signature plotting works great, it plots the point in a recognizable pattern, but the algorithm to compare them leaves much to be desired. It was very difficult making what I have, and I couldn't come up with ways of improving on it. It struggles a lot with

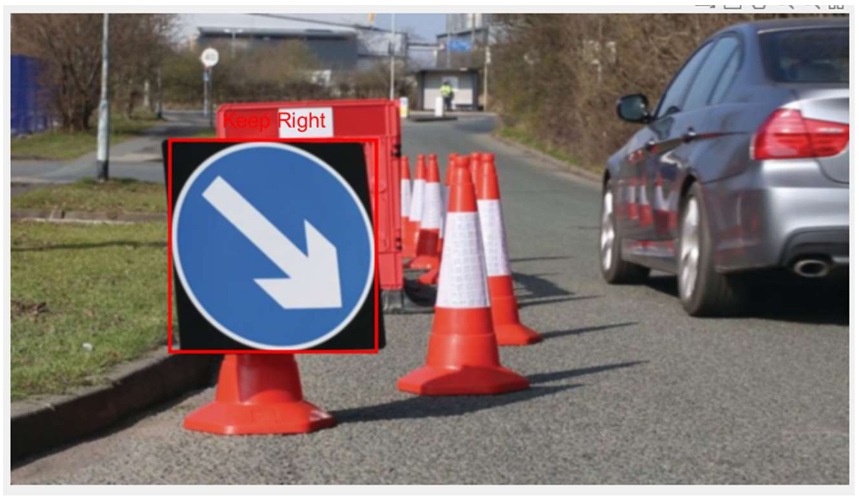
advanced shapes, like distinguishing between a duck and a construction sign. A big problem with the method is that it is computationally heavy, so I had to try and optimize it as much as possible.

The design of the pipeline can be greatly improved. There is a lot of room for improvement in the signature comparison algorithm, which can be improved to improve accuracy and further optimized. On top of that a lot of the variables can be tweaked to maximize accuracy in edge detection and shape recognition. These improvements can be playing around with numbers, until the program has greater accuracy, or replacing them with functions that adapt to the image used. I've used a healthy mix of both, variables and functions, but I see room for improvement. Lastly the image processing part of the pipeline can be vastly improved. Besides changing some of the variables to functions, more operations can be performed, to get more accurate RGB masks. An example can be sharpening, blurring, diluting, or filling the image in the correct regions with the correct values, though from my testing I wasn't able to use those to my advantage.

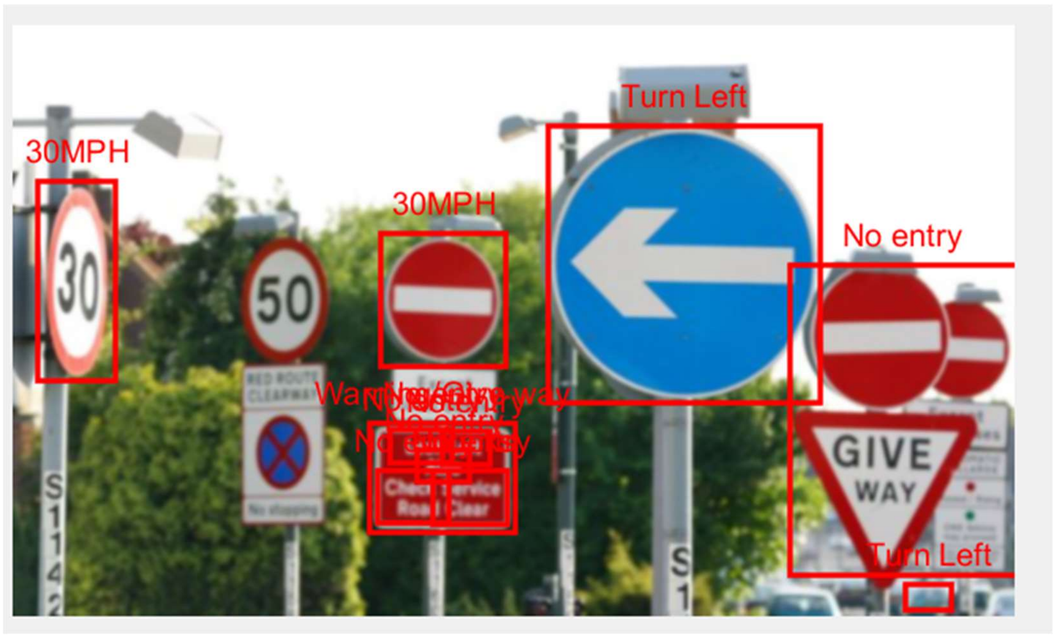
Appendix

1. Correctly detected signs, most signs with simple lighting and not too many colorful objects detect fine.

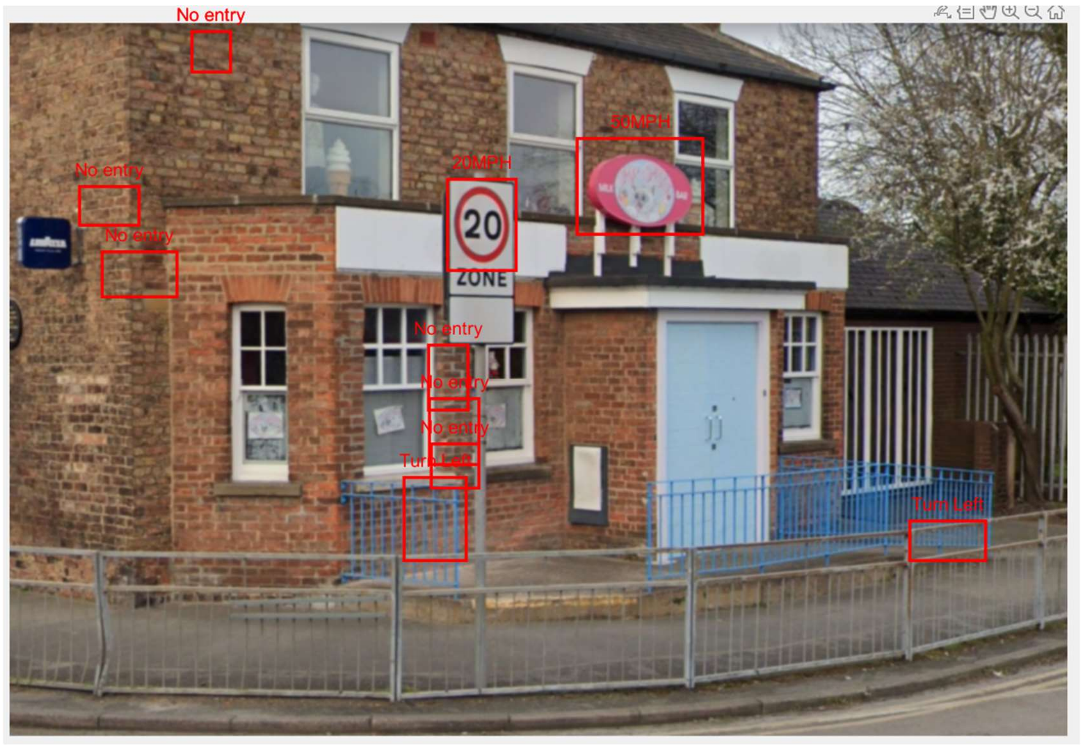




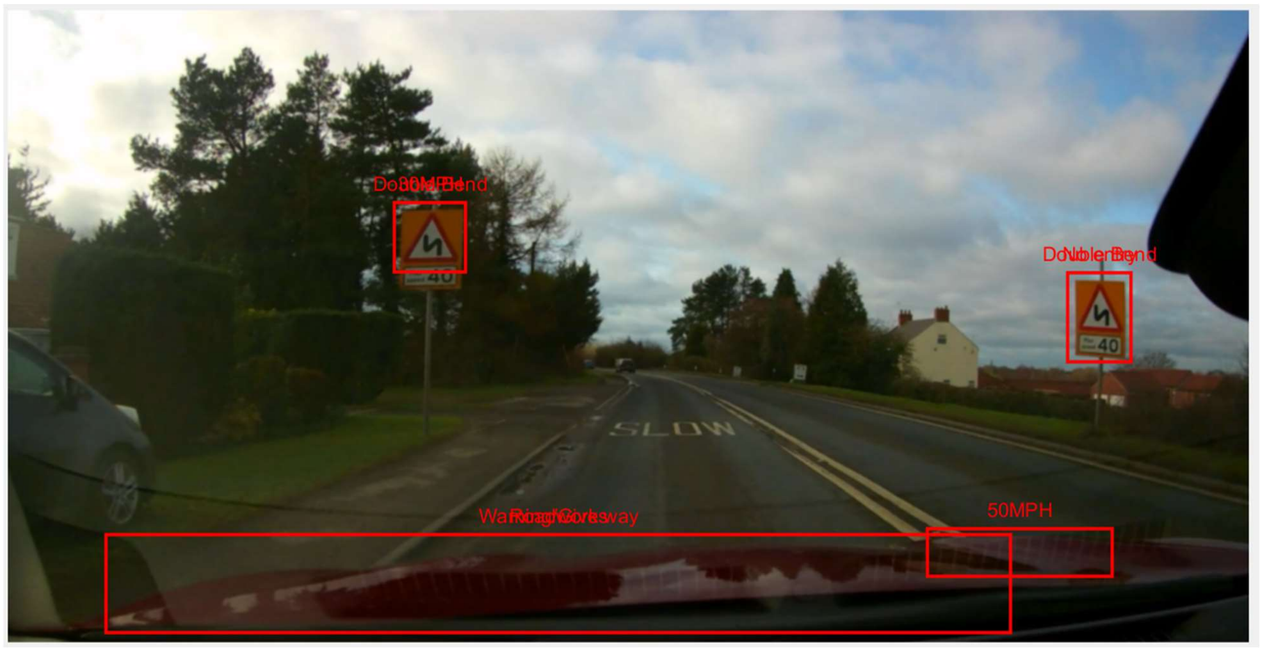
2. The program detects most of the signs, but treats one of the signs as multiple



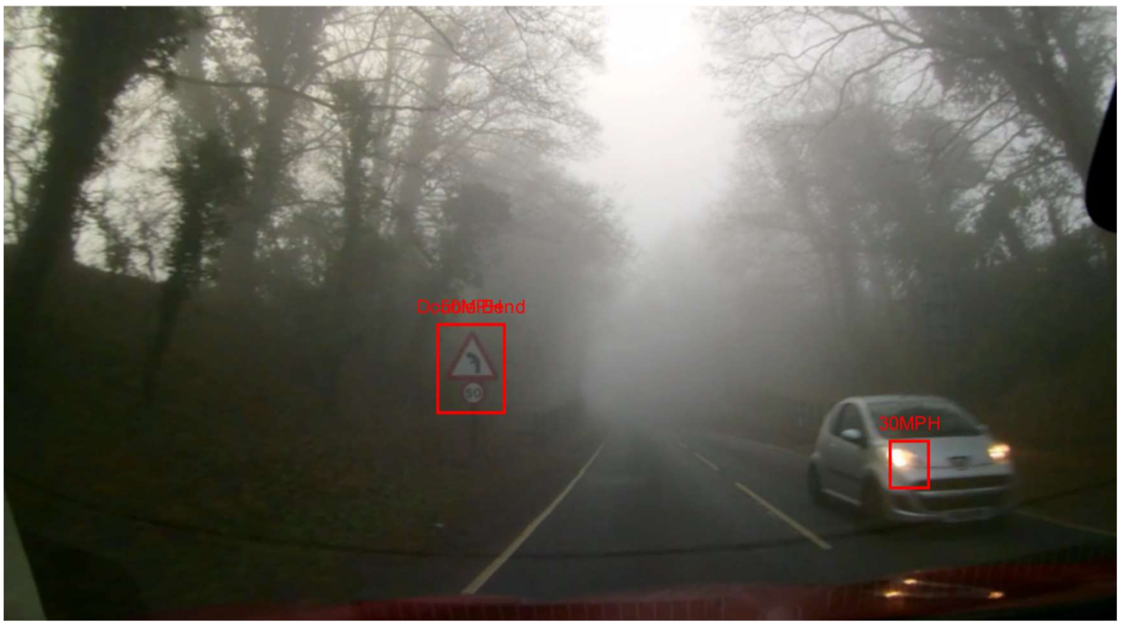
3. Program struggles with the amount of red in the image, detects false signs, but detects the one that is there correctly

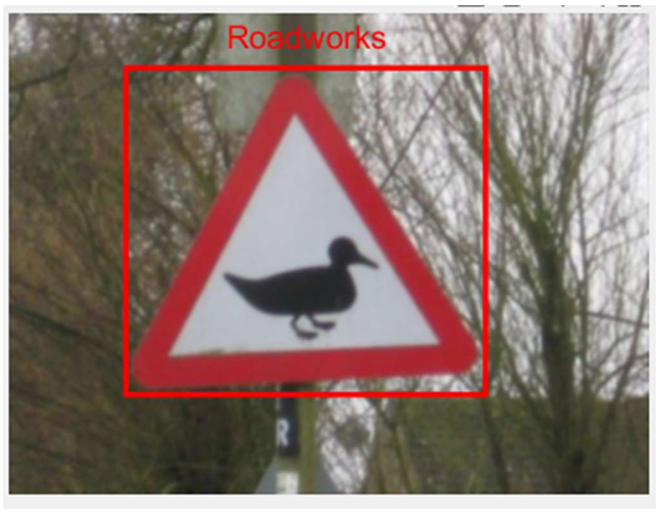


4. camera footage has hood of car with similar color and saturation as signs, resulting in confusion. The program detects both signs correctly, but wasn't formatted to look well with two signs overlapping. (detecting double bends and speed limit, but the number is wrong)



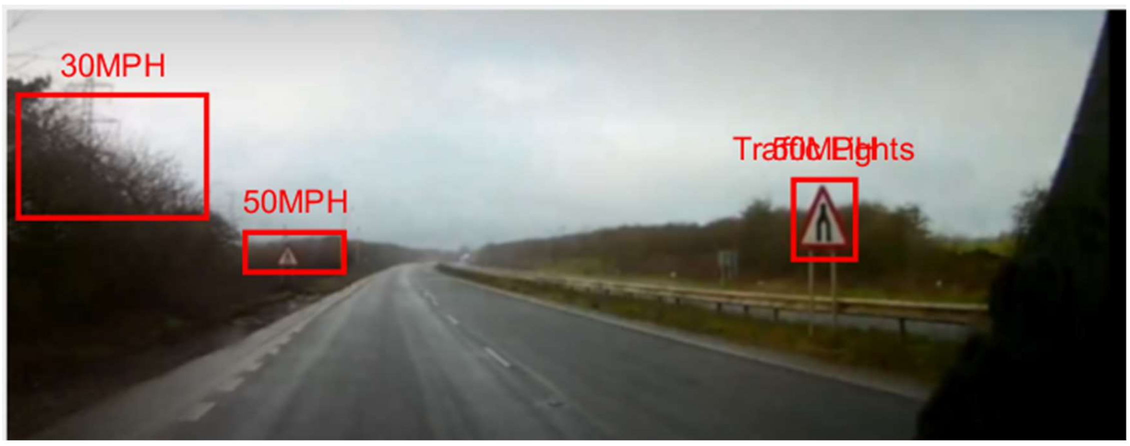
5. Double bend and speed limit detected correctly, but headling of car false detected as a sign, because it's very saturate which fools the program



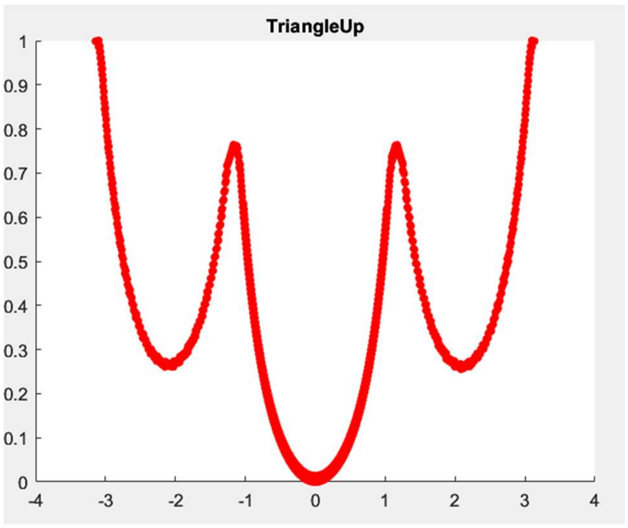


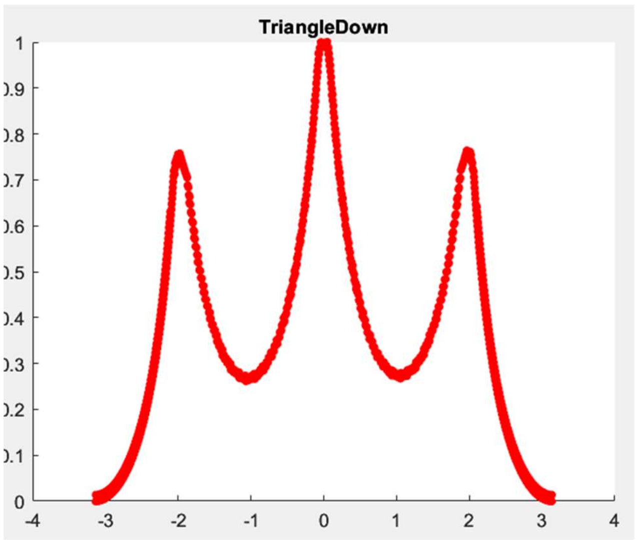
6. Difficult signature reported wrongly as roadworks instead of duck

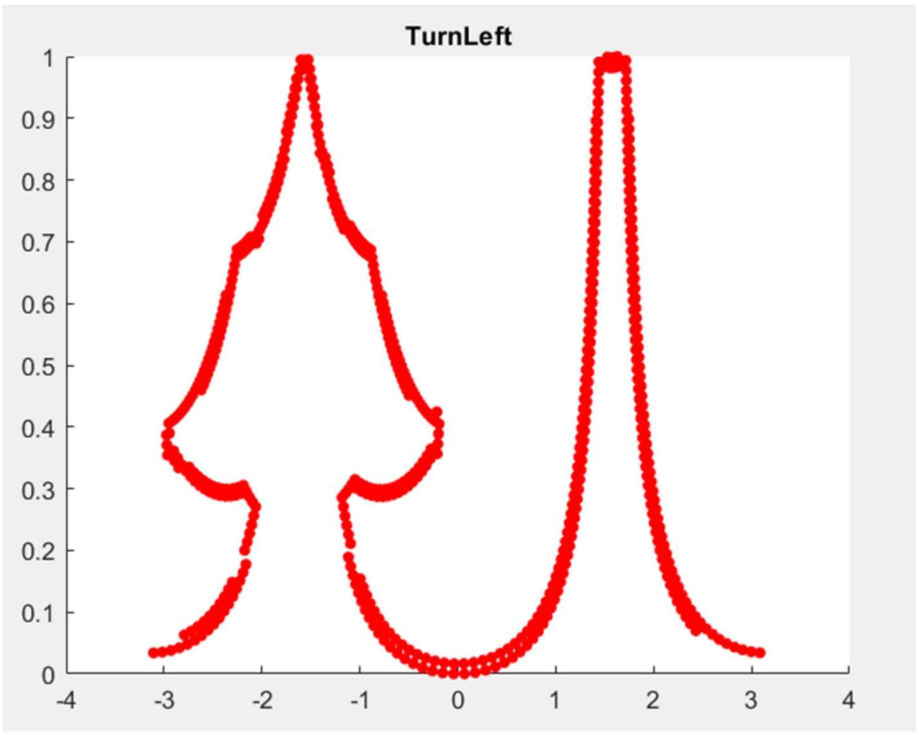
7. Reddish trees detected as sign, signs miss detected, but location accurate

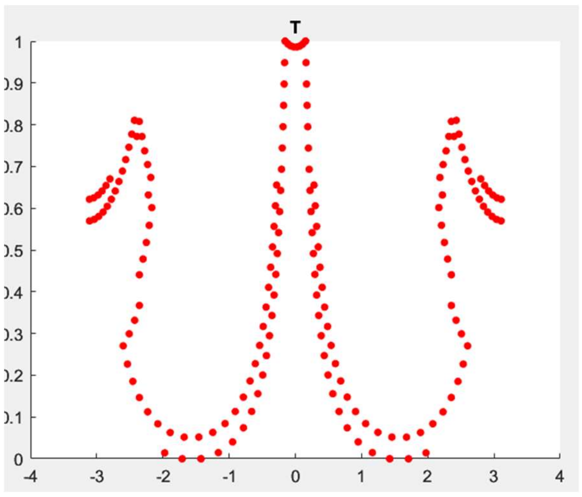


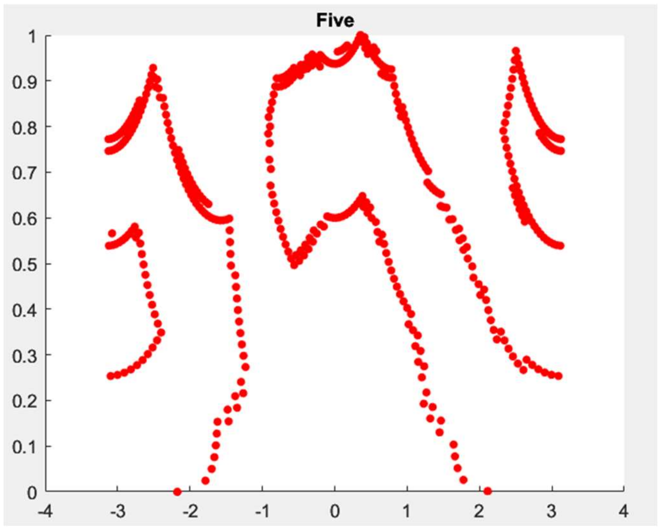
Example angle-distance signatures

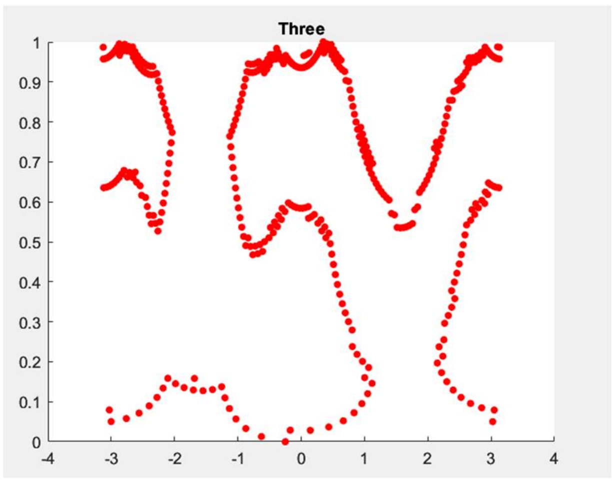


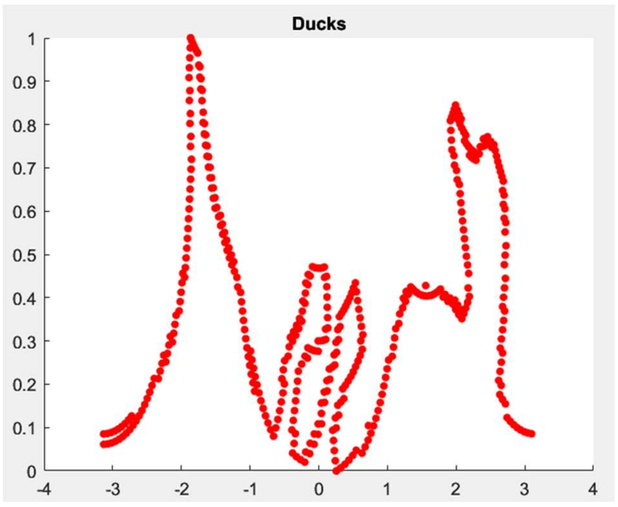





T


Five


Three


Ducks

References

Setiawan, Andika & Yunmar, Rajif. (2020). Implementation of Distance-Angle Signature as Object Shape Identifier for Digital Images. Pixel :Jurnal Ilmiah Komputer Grafis. 13. DOI: 10.51903/pixel.v13i1.158