Automotive Pothole Detection System

Niall Mulcahy
BSc in The Internet of Things
Waterford Institute of Technology
Waterford, Ireland
niallmul97@gmail.com

***Abstract*— this document outlines the design and implementation of a Pothole Detection System, developed using MATLAB. This will be used to detect if there are any potentially unseen potholes one the road that the driver may not be aware of. Two different processes will be tested to identify potential potholes, circle detection and blob detection.**

Keywords - MATLAB, Edge Detection, Circle, Detection, Hough Transformation, Blobs, Noise, ADAS

I. INTRODUCTION

While it is important to constantly be aware of your surroundings while driving, it is essentially impossible to be prepared for every conceivable situation or obstacle.

According to "Engineers Ireland" the average national road quality in accordance with the "International Roughness Index (IRI)" (which is on a scale of 1-10) scored only 4.2. This survey covered 13010 km across 34 towns, counties, and cities, with Waterford City scoring only 3.7[1]. This would seem to imply a near constant presence of potholes on Irish roads. This Pothole Detection System aims to aid drivers by alerting them to potholes of which they may not be aware.

II. IMAGE PREPROCESSING

Before I can even begin to try and detect potholes using either of my methods, firstly the image has to be processed in order to reduce noise for a clearer image. The original coloured image is imported and is then converted to grayscale using the rgb2gray Matlab functionality.

Fig.1 (RGB)

Fig.2 (Grayscale)

"Salt and Pepper" noise is then added to the image and the now noisy image is filtered using a median filter.

Fig.3 (Salt and Pepper noise)

Fig.4 (Median filter for noise)

Worth noting is that the filtered image above actually turned out somewhat blurry. This will have some interesting effects on each method.

From here, the image is converted to a binary image. The theory behind this is that a pothole is, as the name would imply, a hole; which in turn implies added depth. Because of this added depth, potholes in images tend to stand out from the rest of the road because of either shadows or them filling up with water, thus giving them a "shine". As such, using the Matlab function imbinarize with the filtered image and a specified threshold, the image will turn black and white, with the pothole(s) clearly standing out from the rest of the road.

Fig.5 (Binarized image)

Fig.6 (Binarized image inverted)

Note that the threshold used when binarizing the image may differ from image to image, but I obtained relatively consistent results with it set to 0.3 (for the majority of images). Furthermore, I found I achieved better results when using an inverted version of the binarized image, this was accomplished using the imcomplement function.

III. METHOD I: CIRCLE DETECTION

A. Edge Detection

This next step is what one would usually do when trying to detect shapes, that being edge detection using one of the four main methods i.e Sobel, Canny, Roberts, and Prewitt[2]. However, while in my case Canny did obtain the best result of the edge detection types, all of them actually did worse than when the image was left in the inverted binarized state.

B.Circle Detection

The method I used for finding the circles (potholes) in the image was by using the imfindcircles function, which is a method for detecting circular shapes via Hough Transformations. This function allows you to filter out potentially unwanted circles by inputting a minimum and maximum radius size.

C.Results

Admittedly, the circle detection method for finding potholes did not yield a satisfactory result. As stated in the "Edge Detection" section, the results using edge detection actually ended up performing worse than without it. Furthermore, the "effects" from filtering that I alluded to earlier can also be seen here, as the best result overall had neither edge detection nor filtering.

Fig.7 (Result with Canny edge detection and no filtering)

Fig.8 (Result with Canny edge detection and filtering)

Fig. 9 (Result with no edge detection or filtering)

D.Conclusion

I think it's fair to say that this circle detection method failed spectacularly, as even the best result was poor at best. The filter for the circle detection radii that I previously mentioned had little to no effect in a positive manner at least,

as increasing the minimum input from the lowest possible value (5) resulted in no circles appearing and decreasing the maximum would just leave you with the smaller circles (none of which I want).

To me, this says that circle detection is not the right approach. There are a few potential reasons for this. For one, the texture of roads, especially when chipped away like with potholes, is very rough and as such, results in a lot of noise in the image which is difficult to properly filter out without compromising the pothole.

Another reason is that the shape of potholes (especially when seen from the angle of an approaching vehicle) are rarely circular and as seen in the best of the results above, tend to be made of smaller circles joined together, again probably due to the chipped texture.

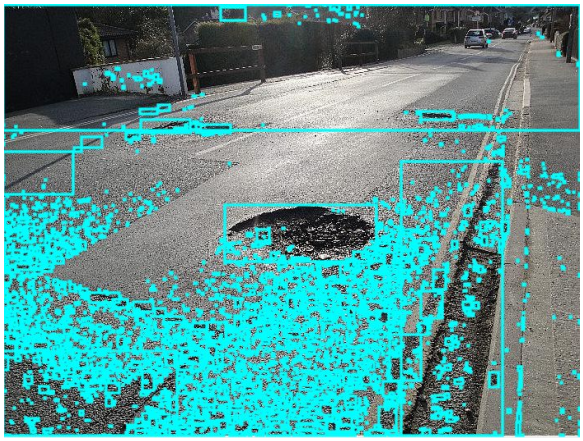
IV. METHOD II: BLOB DETECTION

A. Finding and Counting the Blobs

The method I used in order to find the blobs in the image was the bwlabel Matlab function[3]. This function is used to find connected objects (blobs) in a 2-D (binary) image and can also return the number of these connected objects.

Once found it is important to highlight them on the image. As I discovered in my attempts to use circle detection, the shape of a pothole is almost never a circle,round perhaps, but not a circle, or at least in a binary image and instead made up of smaller circles. So instead, I opted to use a box. To do this, I need the edge data of the blobs. I accomplished this by using the method regionprops with the input "BoundingBox" in order to return the "smallest rectangle containing the region", as well as the coordinates of the top left corner and both the length and width of the box[4]. Loop through the blobs, I can add each blob's BoundingBox, however, without further size filtering, it's far from perfect.

Fig. 10 (Blob detection with no size filter)



B.Filtering the Blobs

Unfortunately, filtering the blobs based on the size of their BoundingBox is somewhat inefficient as it requires me to know the average size of the BoundingBox before actually making the boxes. Obviously (well, to me at least) this cannot be done, so it requires me to loop through the blobs twice, once to get the average size of the box without actually displaying them, and the other to display them based on the desired filter.

There are a few ways by which you could filter by size. Computationally, area probably makes the most sense seeing as length and width are already available, but actually I found that getting the diagonal length (i.e. length from the top left corner to the bottom

right corner) via Pythagoras' Theorem yielded the best results. For the initial loop, I got the diagonal length adding each of them to a list. Then because of the massive quantity of small values, the mean diagonal length was often skewed. A simple loop removing those less than 10 countered this, and from the remaining lengths, the average diagonal lengths were found. Once again I looped back through the loops, this time imposing my filter such that any box with a diagonal length above that of the mean and less than several times the mean (I found 8 to be a consistent value) would be displayed.

C. Results

The results for the blob detection, while not perfect, were far better than those of the circle detection. Interestingly, just as the image filtering had a negative effect on the result for the circle detection, the opposite was true here, where the filtered image yielded far better results than the non filtered image.

Fig 11. (Blob detection with image filtering)

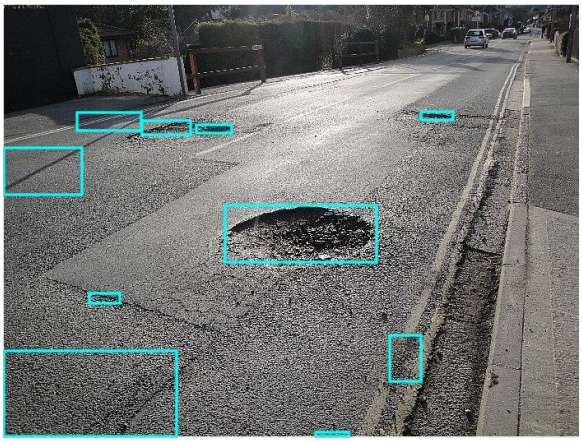
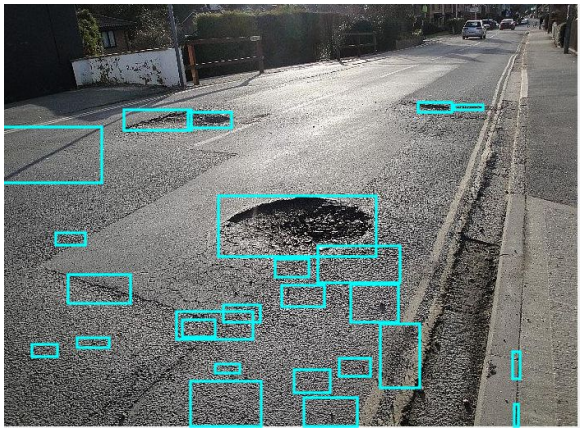


Fig 12. (Blob detection, no image filtering)



These positive results remained true for all of the images that I tested (with some minor tweaks to the binarization threshold, or the usage of the inverted image), for example in Fig. 13 the binary threshold was set to 0.6 (as opposed to 0.3) and the image was inverted.

Fig.13 (Blob detection, increased threshold, inverted)

D. Conclusion

While not perfect, I am overall satisfied with the outcome of the blob detection method as for the most part, it managed to highlight what are clearly potholes, as well as those in the distance that may not be so clear. To me, this is the most important part, as the camera may be able to see the potholes before the driver potentially could and as such, provide ample

warning time. This shows that this has potential as an ADAS feature.

V. POTHOLE DETECTION AS AN ADAS FEATURE

Based on the results of the blob detection, I believe that such a feature could be implemented. As mentioned in the previous section, what is important, is that the camera sees the potholes before the driver necessarily needs to react, so they have the time to prepare to avoid them. As such, I think it would probably be for the best to implement the camera at a height, either on the dash, or better still, on the roof. In either case, the image would most likely have to be cropped as part of the car would be visible from such a perspective. I would suggest cropping the camera feed regardless of position in order to just focus on the road. Cropping was not necessary in my case, as the majority of the images used in testing contained only the road.

Of course, feedback to the driver is still required. One method could be to make use of the reversing camera/screen found in some cars, whereby, when not reversing the screen could be used to display the upcoming road, highlighting the potholes like in the images above. This however would require the user to constantly check this screen and could be potentially hazardous. I believe this in combination with the "beeping" sound produced by reversing/parking assistance features would work. Or perhaps, haptic feedback via the seat or steering wheel, of varying intensity depending on how close the pothole is.

VI. CONCLUSION

The overall goal of this project was the identification of potholes. Now, while my initial effort by means of circle / edge detection was admittedly a failure, my second attempt with blob detection (while not perfect) was relatively successful and as such, I believe that I have achieved that which I set out to do with this project.

VII. REFERENCES

[1]Engineersireland.ie. (2020). [online] Available at: https://www.engineersireland.ie/EngineersIreland/media/SiteMedia/groups/societies/roads-tranport/Assessing-the-Condition-of-the-Irish-Regional-Road-Network-by-Kieran-Feighan-and-Brian-Mulry-28-11-2012.pdf?ext=.pdf [Accessed 2 Jan. 2020].

[2]Samuel, M., Mohamad, M., Saad, S. and Hussein, M. (2018). Development of Edge-Based Lane Detection Algorithm using Image Processing. *JOIV : International Journal on Informatics Visualization*, 2(1), p.19.Available: https://www.researchgate.net/publication/323198429_Devel opment_of_EdgeBased_Lane_Detection_Algorithm_using_Image_Processin g. [Accessed 2 Jan 2020].

[3]Uk.mathworks.com. (2020). *Label connected components in 2-D binary image - MATLAB bwlabel- MathWorks United Kingdom*. [online] Available at: https://uk.mathworks.com/help/images/ref/bwlabel.html [Accessed 2 Jan. 2020].

[4]Uk.mathworks.com. (2020). *Measure properties of image regions - MATLAB regionprops- MathWorks United Kingdom*. [online] Available at: https://uk.mathworks.com/help/images/ref/regionprops.html [Accessed 2 Jan. 2020].