TEXT DETECTION IN SIGNAGE
Improving OCR Accuracy

Stephen Moore

Student Number: 900618883

DSP Term Project Technical Report Spring 2020

4/30/2020

ABSTRACT

Optical character recognition is a widespread technology, useful for recognizing characters within images, and attempting to make meaningful words out of them. Many OCR applications exist, but an exercise in improving one proves useful to learning digital signal processing. The problem I found is the MatLab computer vison toolbox's OCR method was rather weak, so I attempted to improve it via pre-processing, utilizing maximally stable extremal regions, holding those regions within bounding boxes, and applying OCR to the boxes. This system has a solid accuracy with natural images with signs, though it could be improved greatly utilizing machine learning methods, namely a convoluted neural network.

I. INTRODUCTION

The question of computers being able to 'see' something has always been one of great interest to me. How a machine can see characters on say, a sign, and then be able to output the text to a user is fascinating. In my research before this assignment I was reading about optical character recognition, or OCR. Basically, it is a technology that recognizes text within a digital image. [1] There's a multitude of OCR algorithms out there, each with their own strengths. Such as Tesseract, Google Cloud Vision, Microsoft's Computer Vison, the list goes on. However, I wanted to see what the version included in MatLab's own computer vision toolbox had to offer.

Often it would read the text of an image, however it would be extremely noisy and be full of miscellaneous characters. Meaning the 'accuracy' was low, both on a character and word level. So, the problem became how can one improve the accuracy of an OCR algorithm? Some things to start, image processing is a huge one as the contrast and character definition on top of the reduction of noise can help immensely. [2] A lot of people will box off an area for the OCR algorithm to focus on as well. Seeking to make this more than a one size fits all project, I researched maximally stable extremal regions. These can be utilized to created bounding boxes where the algorithm can then operate reasonably well on most images you throw at it. For this report I will be using

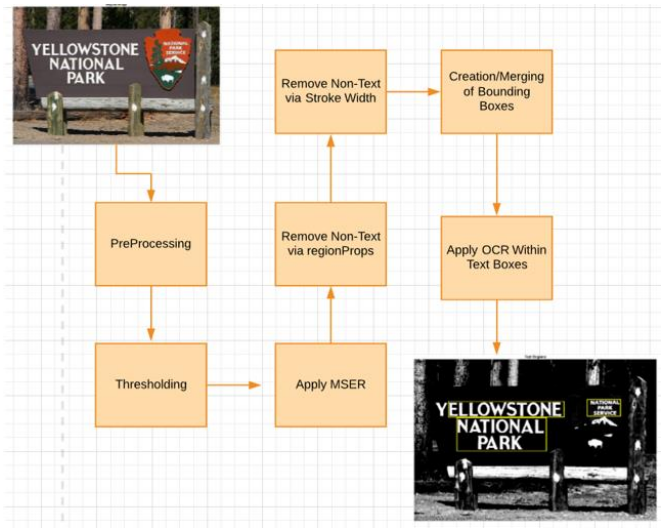

Fig 1: Methodology

For this project I will be using the above methodology in stages. Starting with the base rgb color image pre-processing is applied, followed by thresholding. This will be covered in Section II. Following this we will apply the maximally stable extremal regions algorithm, then improve it via geometric region props, and lastly calculate stroke width distance which will allow the removal of regions that do not have similarity to characters. This will be in Section III. In Section IV the application of bounding boxes to aid OCR will be discussed. Lastly Section V will conclude the paper and discuss future work that would be possible with such methods.

II. PRE-PROCESSING

Fig 2: Original Images

Pre-processing is the use of algorithms in order to preform image processing on digital images. It is a necessary step in computer vision, and in itself a subfield of digital signal processing. An image is a two-dimensional array of numbers ranging between 0 and 255 [4]. Some possible steps of pre-processing would be to read the image, change hue/saturation values, segmentation via thresholding, and so on.

Using the original images in Fig. 2, I used rgb2gray to convert the truecolor rgb image to a grayscale image. This function eliminates the hue and saturation, while retaining the luminance of the image. [3] The grayscale image is a good step, but we must increase the contrast of the image. In doing so we are defining edges, which works in our favor as characters already have defined edges and will come out stronger. To do this we will use imadjust, setting the limits of the contrast in so that we don't ruin the image.

From here we would want to remove as much noise a possible. I will be using median filter pre-processing, with the function medfilt2. 2D median filtering is useful as it will preserve the edges within an image, while also removing noise. This is useful in situations with trees or bricks where there's a lot of noise present. This is a commonly used pre-processing step. Below in Fig. 2 are the images after the steps of pre-processing. The characters are more defined, and the background areas are slightly clearer of noise.

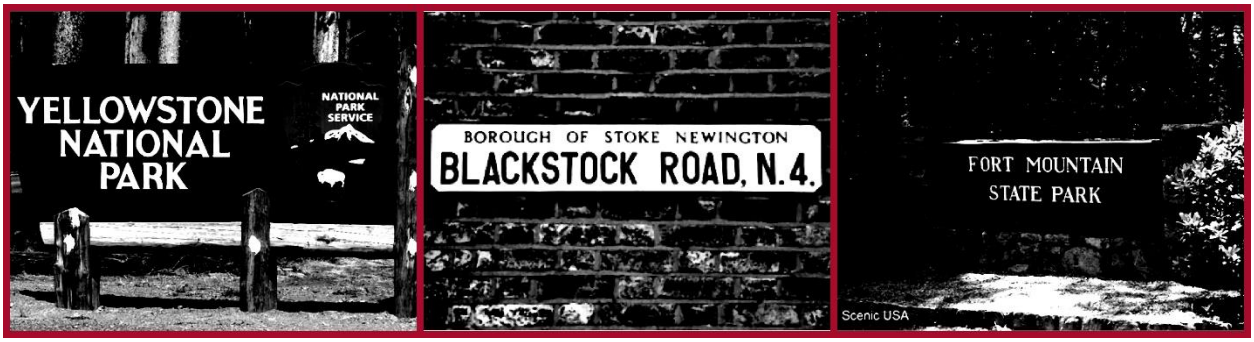

Fig 3: Images After Pre-Processing

The last step taken with image processing is to implement segmentation via thresholding. Thresholding is a very simple, yet effective method of image pre-processing. Using imbinarize, we create a binary image from a greyscale image by setting all the values above a globally determined threshold to 1s and setting all values below the global threshold to 0s. The threshold was default, but I accounted for foreground darkening for signs that have a darker color like the Fort Mountain sign in Fig. 3.

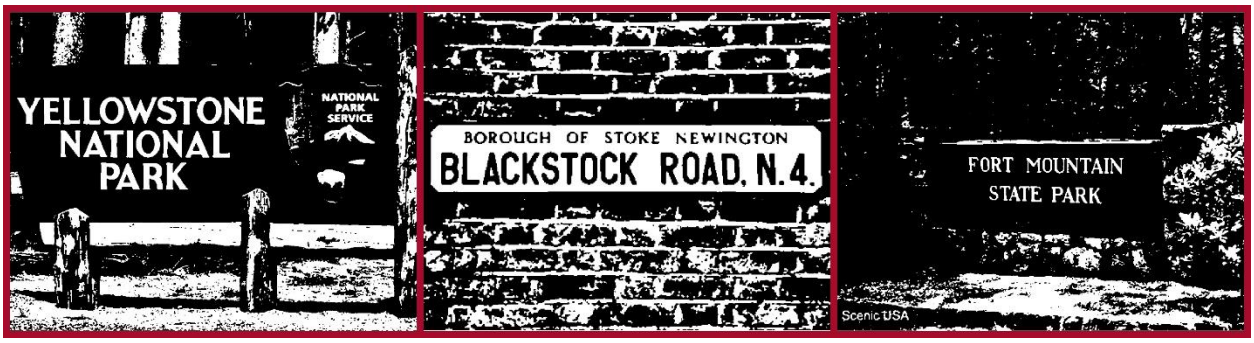

Fig 4: Images After Segmentation

In Fig. 4. We can see the results of the image segmentation. The binary images we are left with after pre-processing are binary, have distinct edges on their characters, and will prove quite useful for the next step. It is important to note that while the pre-processing works well on most images, this is not something that will work on everything due to the nature of the values of each algorithm, this will be discussed later.

III. MAXIMALLY STABLE EXTREMAL REGIONS

Maximally stable extremal regions, or MSER is a method used to detect blobs in images. The idea is that we will end up with ‘stable’ regions after we pass the image through a varied range of thresholds. The ‘extremal’ regions are the areas with higher (brighter extremal) or lower (darker extremal). [5] Having performed the pre-processing, this will yield better results as the regions are more polarized. We will end up with more regions that a non-processed image, but these regions will be easier to remove with the following methods. Below in Fig. 5 are the test images with applied MSER. The highlighted areas are the stable regions.

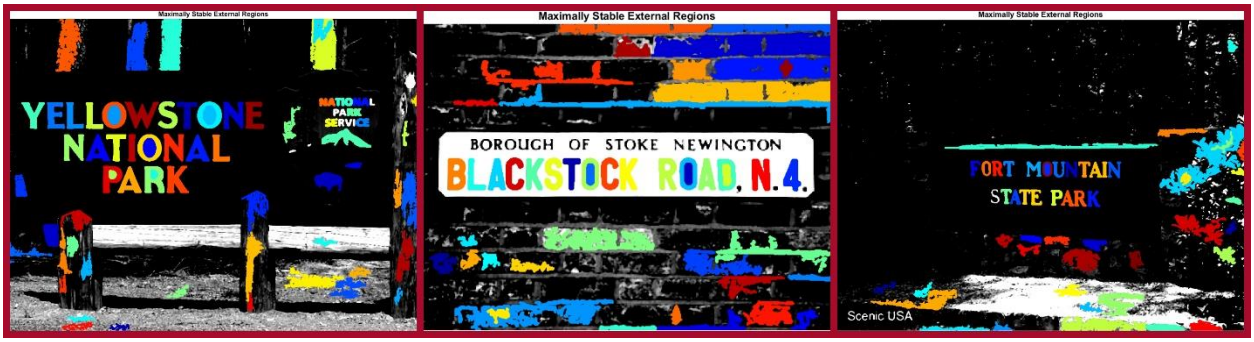

Fig 5: MSER Images

We must begin the process of removing non-text areas so that we can single out the characters we'd like to preform OCR on. The first method to do this will be utilizing regionprops shape measurements in order to filter out these unwanted regions. Some properties we will utilize include eccentricity, solidity, extent, and degree. An example being eccentricity, where 0 = a circle and 1 = a line. This is useful as characters have a rather similar shape in most typefaces. After setting these values, we locate the regions they associate with then delete them from our MSERs to leave us with predominantly text regions. It is important to note that these are all values that are set by the user, and therefore will not fit each and every image. Fig. 6 displays the outcome of this method, with mostly characters left over.

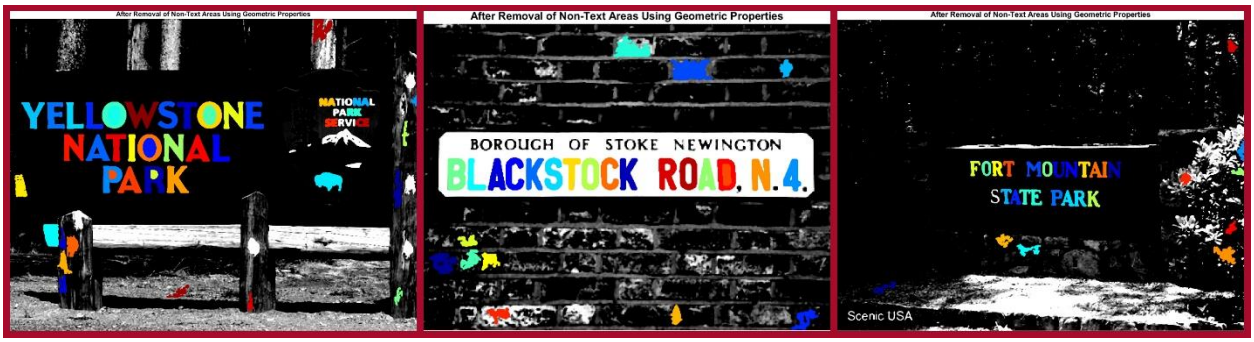

Fig 6: MERS After Applying Regionprops Removal

The last step of cleaning up MSERs is to first calculate the stroke width variation, then to remove regions based on this. What we are looking for is uniformity. We select a MSER then calculate the distance between its pixels in both the lines and curved areas of the selected region, then we transform it down to a pixel to then look for uniformity. As stated earlier, characters have a general uniformity about them. We can remove the values that do not contain said uniformity, which should, in theory, leave us with only text regions and a couple of missed areas. Fig 7. Shows examples of a uniform region (the letter O), and two non-uniform regions that would be removed.

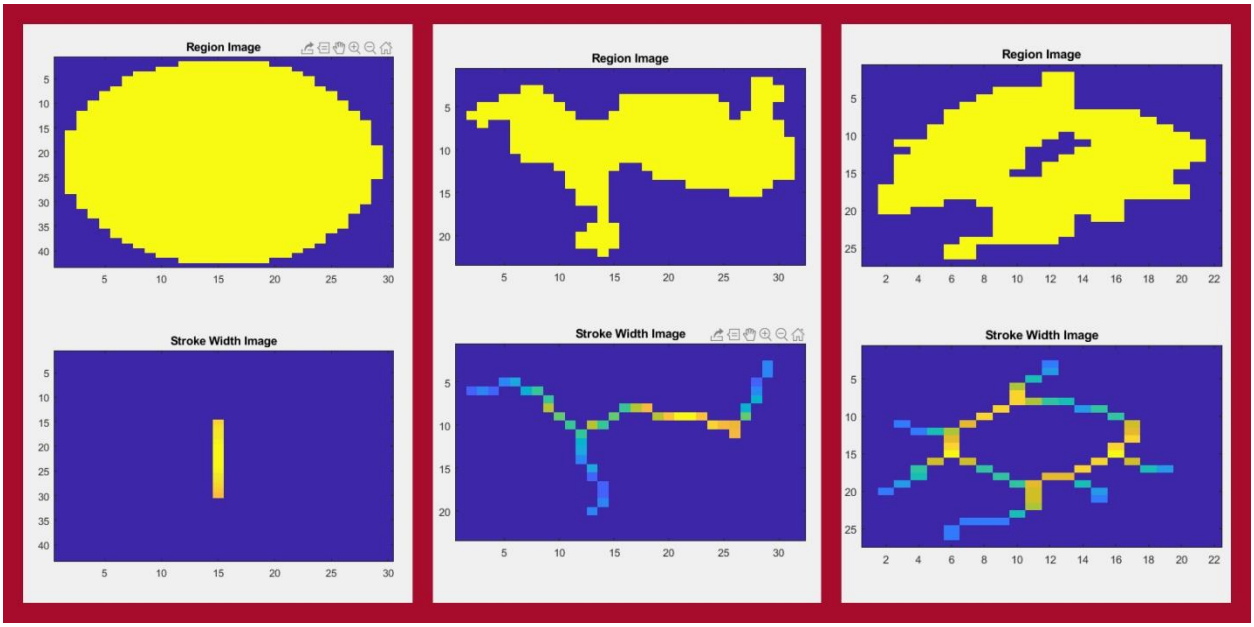

Fig 7: Stroke Width Calculation

After removing these non-uniform regions, we are left with what we can see in Fig. 8. From here we will take these regions and add bounding boxes in order to crate areas in which we can direct OCR to operate. There are areas outside of the characters that were not successfully removed, but these can be handled in the next steps.

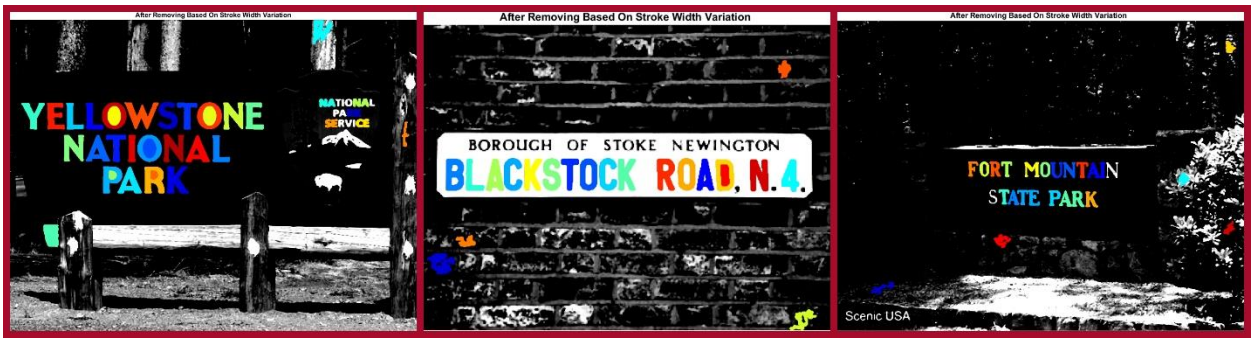

Fig 8: MSER After Removal of Non-Uniform Regions

IV. IMPLEMENTATION OF OCR

As stated above, the next step is to create bounding boxes using the MSERs that we had spent time cleaning up in the last section. Using boundingbox we can return the smallest rectangle enclosing a polyshape, these shapes being out MSERs. As can be seen in Fig. 9, there are a few stray areas.

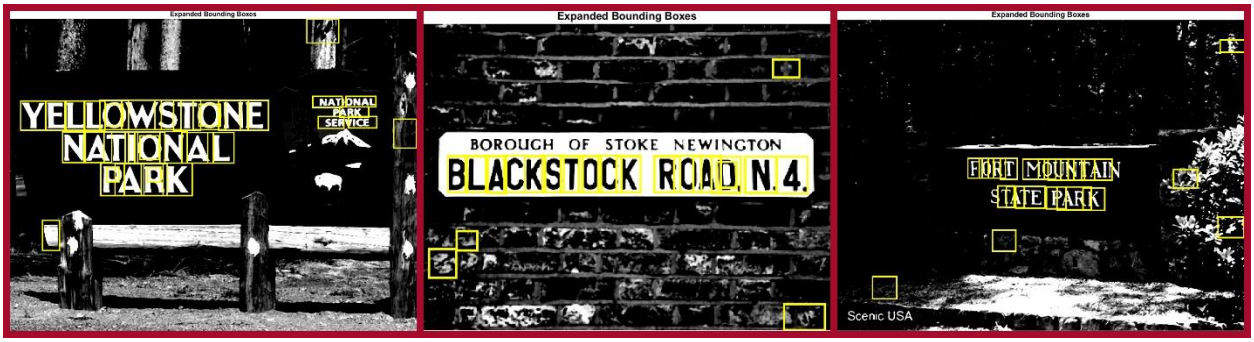

Fig 9: MSER With Bounding Boxes

To get rid of these areas, we will use bboxOverlapRatio to combine these boxes into larger regions. However, this can be a double-edged sword as while it gets rid of stray regions away from the text areas, it will drop characters that are not close enough to the rest of the group. To provide an example of this, Fig. 10 displays a fast food sign with very spaced out characters. In this case, most of the image is dropped from the bounding boxes, not leaving us with much to work with.

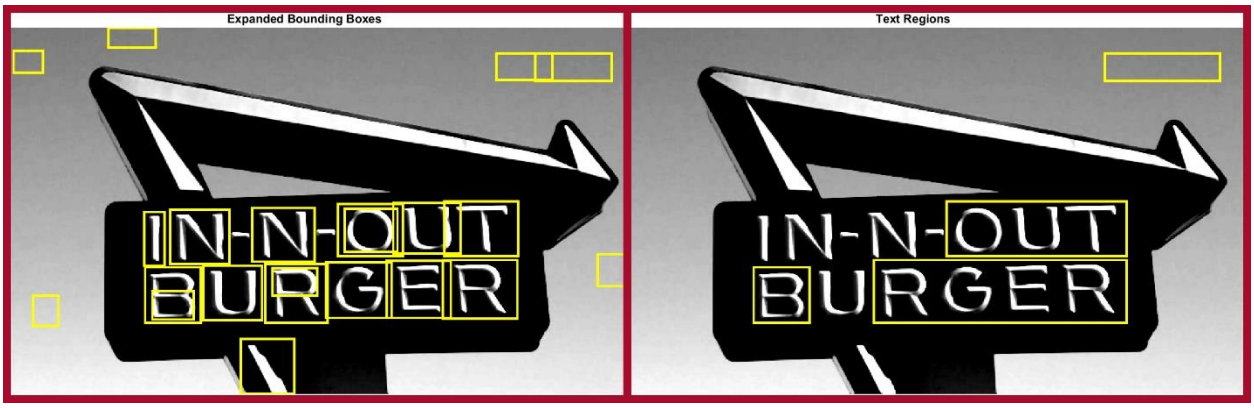

Fig 10: Overlap Failure

Regardless from cases such as the above. Most images will work with the program. Again, a lot of the values are entered by the user, so in the case of the fast food sign we can make the bounding boxes bigger and the overlap area larger, but this would harm other results. After we've completed the above steps, applying OCR with an English and numerical character set will yield the following results on our three test images. You will notice the dropped characters, but overall a majority of the text was recognized, improving the accuracy of OCR. See Fig. 11 for the results.

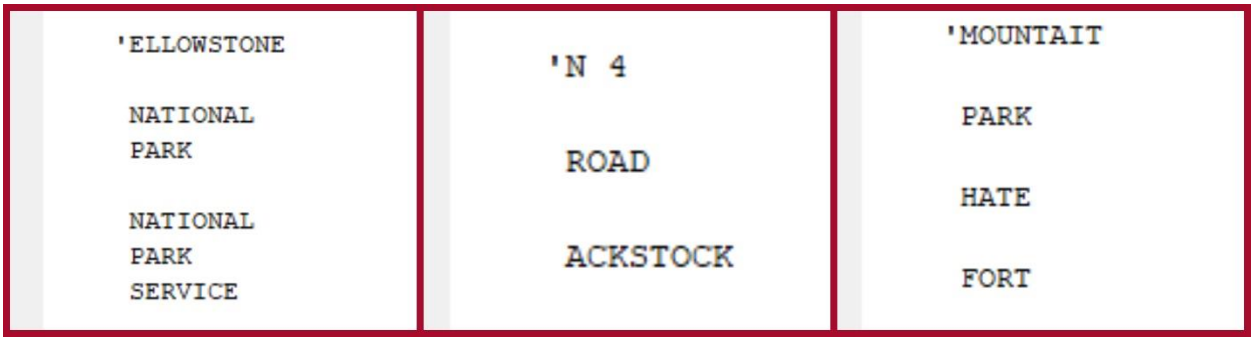

Fig 11: Results of OCR after Pre-Processing and MSER Application

V. CONCLUSION

In conclusion, we can see that not only can OCR be improved, that there is still much more to do to achieve the highest possible accuracy. Computer vision can most definitely find success in real world scenarios by testing on natural images, versus close up images of characters like license plates or credit cards. While those are still useful, there is a lot more noise in images like these. Detecting where useful text is and recognizing those characters into meaningful words or sentences could have many applications. Perhaps something similar could be used in self-driving cars? Interpreting road signs might be a useful tool for that.

However, for this to happen this method would need improvement. Signal processing can be greatly enhanced via machine learning methods. In this case a convolutional neural network could be utilized to improve the accuracy, and we already have the feature extraction methods ready. Many things could be improved such as the issue that was discovered regarding character spacing and the dropping of said characters.

Regardless, this was as very entertaining exercise with digital signal processing, and I look forward to taking the time to improve upon it in the future.

REFERENCES

[1] Christensson, Per. "OCR Definition." TechTerms. Sharpened Productions, 16 April 2018. Web. 04 May 2020. https://techterms.com/definition/ocr.

[2] Harris, Joshua, and Fabio. 2020. “Improve OCR Accuracy With Advanced Image Preprocessing.” Document Data Capture And Workflow Automation. April 28. https://docparser.com/blog/improve-ocr-accuracy/.

[3] “Documentation.” 2020. MATLAB Documentation. Accessed May 5. https://www.mathworks.com/help/index.html?s_tid=CRUX_lftnav.

[4] Canuma, Prince. 2018. “Image Pre-Processing.” Medium. Towards Data Science. October 11. https://towardsdatascience.com/image-pre-processing-c1aec0be3edf.

[5] “MSER (Maximally Stable Extremal Regions).” 2020. U.S. Army Mission and Installation Contracting Command. Accessed May 2. http://www.micc.unifi.it/delbimbo/wp-content/uploads/2011/03/slide_corso/A34%20MSER.pdf.

[6] “Computer Vision.” 2020. Wikipedia. Wikimedia Foundation. April 28. https://en.wikipedia.org/wiki/Computer_vision.

[7] “Deep Learning-Based Text Detection and Recognition in ...” 2020. Accessed May 5. https://medium.com/brightlab-techblog/deep-learning-based-text-detection-and-recognition-in-research-lab-bb3d61797f16.