3D Print Quality Estimation

Hunter Obendorfer 1834106

University of Houston

COSC 4397 – Computer Vision

Table of Contents

Introduction

As 3D printers become cheaper and more accessible, more people will try their hand at 3D printing. Starting out in 3D printing can be difficult, there are many variables that must be controlled to achieve high dimensional accuracy, no discolorations, and no elephant's foot (where the base of the print is wider than it is supposed to be). There are other qualities that can be seen visually but the implemented functions focus on the three mentioned. These functions aim to help new users by finding common mistakes in the 3D printed object and linking them to guides that they can follow to remedy the errors made. Doing so will hopefully let those new to the hobby become proficient in 3D printing faster by showing them what to look for and how to fix errors. The functions implemented take images and do various calculations to find the errors listed previously, these functions take in a very specific format for the images as in it their current state, the functions are a proof of concept rather than functions ready to be used in a phone app.

Dataset Description

The dataset is divided into 3 categories, burns, dimension estimates, and elephant's foot. Every 3D printed object in the dataset was printed by the author. To expedite the development of the functions, many restrictions were put into place for the data. First, the objects must be light in color and against a dark background. Second, to maximize accuracy without having to do sub-pixel calculations, high resolution images were used (4608x2592) taken with a Canon PowerShot SX530HS camera, many images use the macro focus option with ISO at 3200, 1/50 exposure time, and a f-stop time of f/5. More specific restrictions were used within the categories of images. The burns category images should contain a burn somewhere on the printed object.

Images used to estimate the size of an object must have the calibration cube (see Determine the Size of an Object in Methodology) be the rightmost object in the image, many objects may be present in the image otherwise. For images used to determine elephant's foot, the object must be at least partially on the right half as well as the top and bottom of the image. The elephant's foot detector will allow for blank background above the image, as not all images will be so close that the object continues off the top of the image. The objects used here must be rectangular.

Methodology

i) Preparing an Image

Images are prepared with the function called 'prepImg(img)' where img is the regular color image taken by a user. The function converts the image to grayscale, applies a gaussian blur with kernel size of 13, then sharpens the edges of the image, and finally normalizes the grayscale image to be of values between 0-255 inclusive. This preparation function returns the prepared image to be used by later functions.

ii) Finding Burnt Spots on an Object

The function 'findBurntSpot(img, objectColor = "white", retData=False)' takes an image (img) and after preparing the image finds burnt spots on the object in the image. It does this by applying a custom threshold. Since only objects that were made of white printer filament were used, after preparing the image, many values in the image matrix were on the extreme ends. The values were near 0 or 255, so the implemented custom threshold would threshold to a range of values, typically 215-219 inclusive, anything outside of that range was made 0 and those inside were made 255. Interestingly, this threshold made a half decent edge detector, it is speculated

that this is due to shadows coming off the object and the restricted nature of the content of the

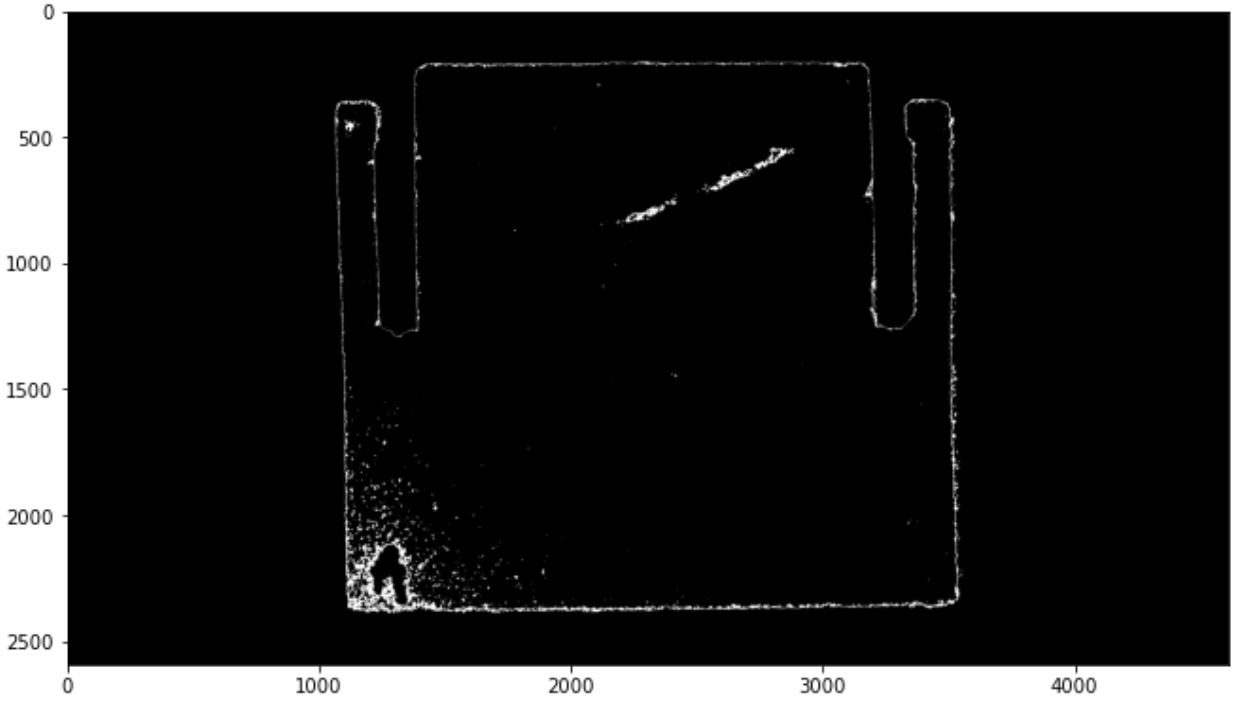

Figure 1: result of custom threshold on Burns image 00.JPG

images used, see Figure 1. After applying the custom threshold, a technique called connected component analysis was used to find blotches in the image [1]. Since burns appear only slightly darker than the object, the burns were retained as blotches in the image. After finding the locations of the blotches, the function returns the blotches as opencv contours, this allows the image to be reused for other forms of analysis, typically, the contours will then have a circle drawn around them after the function is called which must be done by the user at this point in time. If 'retData' is set to true, the operation is silent and only returns the contours list and does not tell the user if burns were found, this is used in the testing code. Currently, changing 'objectColor' does nothing as this is not yet implemented.

iii) Determining the Size of an Object

Inspiration for writing this function came from the old TV show Mythbusters, on occasion, they would use a slow-motion camera to measure the speed of projectiles. The projectile would pass in front of a background of black and white bars of known size. The method was chosen because it would be easier for a user to print a cube than to calibrate a camera to determine the size of an object. This function uses an object of known size to measure the size of other objects. The function 'determineSize(img, width, height, error=5.0, retData=False)' takes an image (img) as its input as well as the nominal width and height of the object. The nominal width and height are the width and height that the object is supposed to be in millimeters. The nominal width and height can be obtained easily in common slicer software such as Ultimaker Cura. After preparing the image, this function uses Otsu's thresholding to get

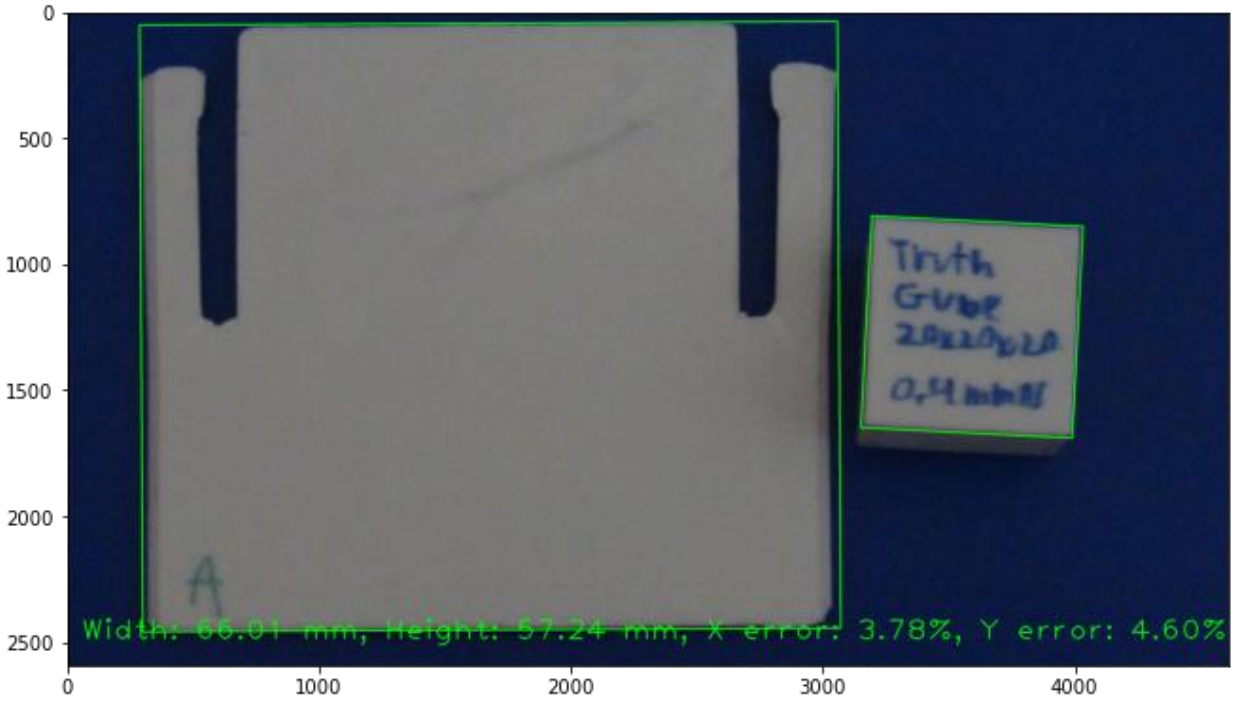

Figure 2: example output for determineSize on Dimension Estimage image 00.JPG

a thresholded image, this technique selects the threshold value automatically [2]. Using the same

contours technique, the minimum enclosing rectangle is found, this rectangle is rotated as the orientation of objects is not perfect and doing so reduces the error calculated. After finding the rectangles, the rightmost rectangle is found and the x component of it in pixels is then mapped to 20 millimeters, it is assumed that the rightmost object is the calibration cube that is a 20 mm cube. Only the x component of the pixels is used as the y component is not as robust as seen in figure 6. This number of pixels for 20 mm is then used to measure the other object's height and width to give an estimate of the width and height. The measured height and width are used to calculate the error against the nominal height and width. These values are then printed on the bottom of the image.

$$Error = \frac{|measured\ value - nominal\ value|}{nominal\ value} * 100$$

If 'retData' is set to true, only the measured width and height are returned in a tuple in that order. Otherwise, the function will return the image as well as tell the user to recalibrate their machine if the error is greater than 5.0%.

iv) Finding Elephant's Foot

Elephant's foot is the result of the printer's nozzle being too close to the print bed, causing the extruded filament to be forced out wider than intended. This causes the bottom of the print near the print bed to be wider than what it was supposed to be. The function 'findElephantFoot(img, error=5.0, retData-False)' takes an image (img) and cuts it in half vertically after preparing it, then uses Otsu's thresholding to get a black and white image of the right half of the object [3]. The function takes this black and white objects and counts the number of white pixels in each row of the top half of the threshold image, the mathematical 1st quadrant if the original image were a cartesian coordinate system centered on the center pixel. Then the

average number of white pixels per row in the top half of the threshold image is calculated. This average is then compared to the number of pixels in each row in the bottom half of the threshold image, if the error is greater than 5.0%, it is assumed that the object has elephant's foot. The user is notified, and a link is provided to show the user how to level a print bed, while the author does recommend the use of feeler gauges instead of paper to do so as the gauges are more consistent. This function does not output an image as elephant's foot can only occur in a single spot, the bottom of the print, minor elephant's foot may be difficult to see sometimes but telling the user that it is present should prompt them to look closely at their object.

Results

i) Results of Finding Burns

The function that finds burns tends to have many false positives due to the nature of the thresholding implemented. The edges of the object are retained in the image and the contours are processed for them, so the function often triggers and will circle the edges. It can detect blob like

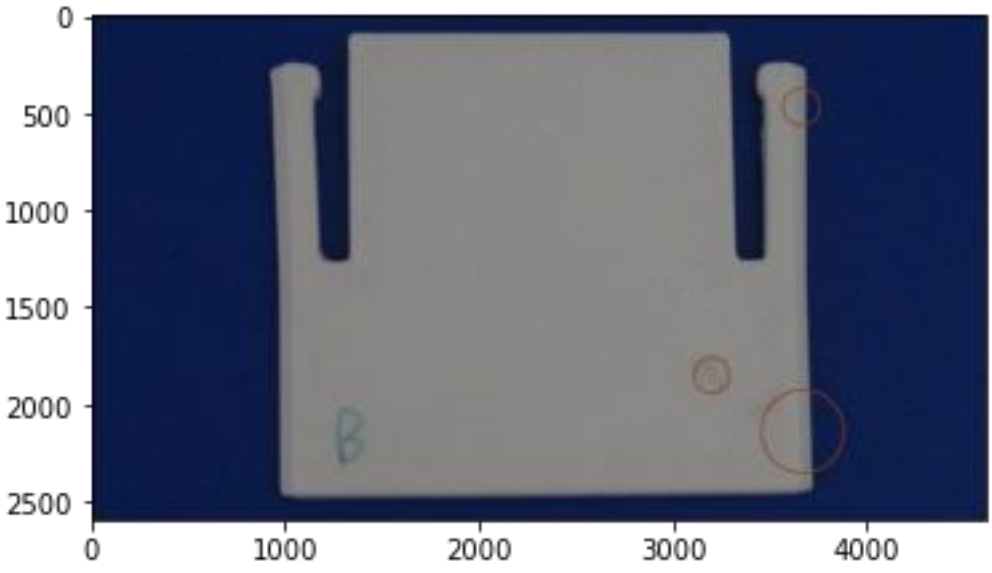

Figure 3: Result of finding burns on Burns image 01.JPG, note that the edges are circled in 2 places on the right, the leftmost circle is a burn.

burns on objects, but it cannot detect line like burns on objects reliably. This function needs improvement to reduce these false positives, possibly by looking inside an enclosing rectangle only and ignoring values within a certain distance from the rectangle.

ii) Results of Dimension Estimation

The estimation of dimensions of an object is relatively accurate. For objects of dissimilar size from the calibration cube, the user can expect an error less than 5.0% when the height and width of the object is found correctly as seen in figure 2. Orientation can affect the error calculation; this is shown in figure 4. Objects similar in size to the calibration cube, ±5 mm, have

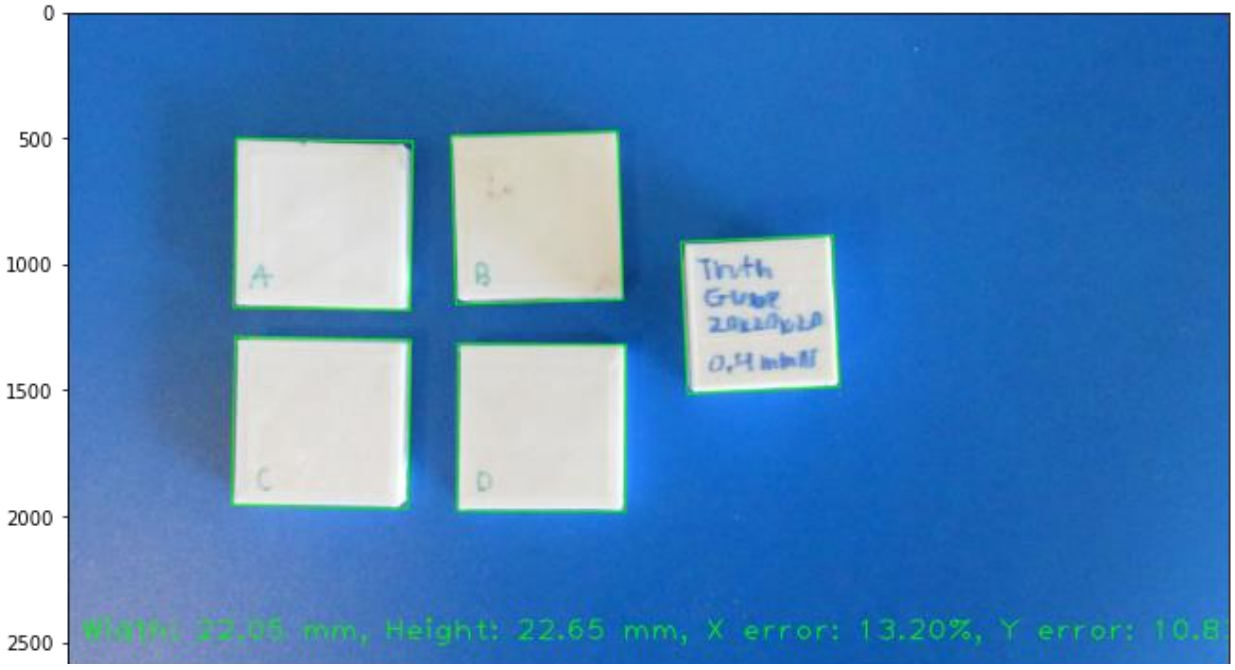

Figure 4: objects in this image are 1-inch squares, or 25.4x25.4 mm nominally.

a higher error, typically 10% or more. This increase in error could be due to the objects taking up less space in the image than others, leading to smaller disparity between the cube and other objects, note that this is not disparity as used in stereo vision. To further decrease error in the future, the user could be able to enter the measured dimensions of their calibration cube, as an uncalibrated 3D printer may not print a perfect cube and the dimensions will be incorrect. As the

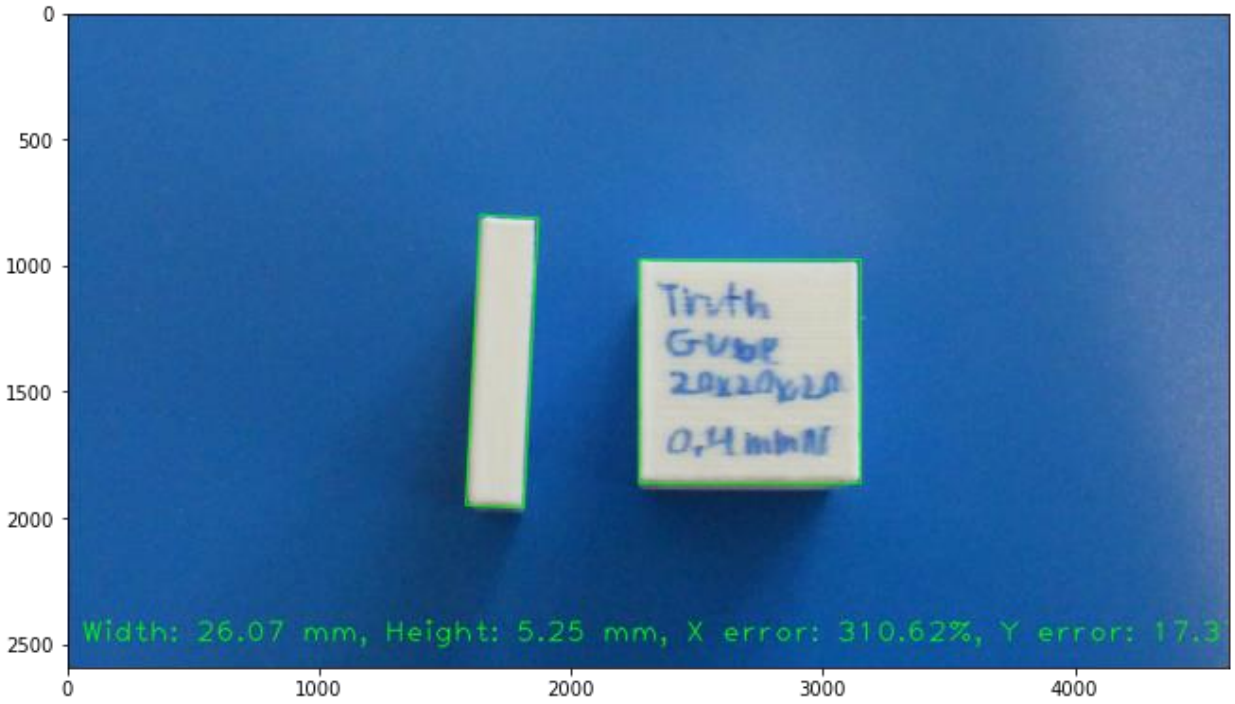

Figure 5: Height and Width were swapped by opencv, causing a large error.

disparity between the calibration cube and measured object increases, the error decreases as shown in figure 6. Overall, low error can be achieved by taking a good picture which is zoomed

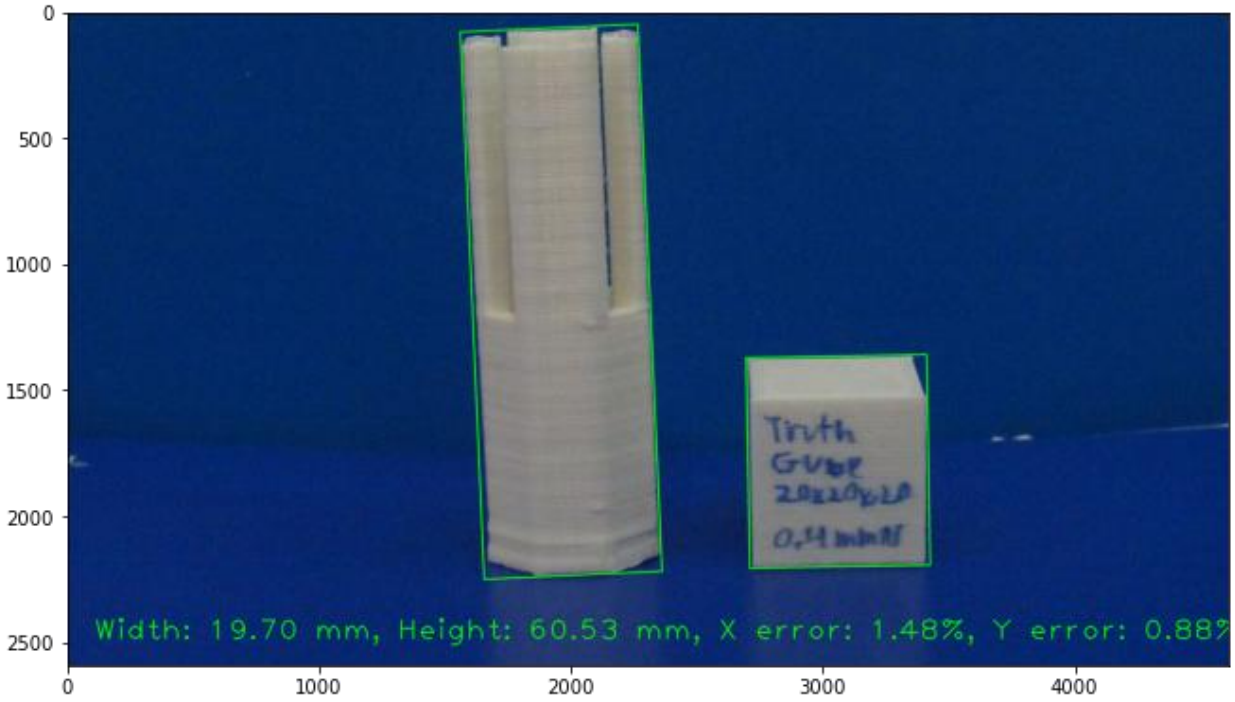

Figure 6: Large disparity in size between the two objects with less error.

in as close as possible, an angle such that the x component of the calibration cube is at its minimum, and the object oriented correctly.

iii) Results of Finding Elephant's Foot

This simple function had the least error, as long as the image being used meets the prerequisites for being used in this function. The image needed to have the object roughly centered and only have a singular, rectangular object in the image. The function will correctly determine elephant's foot if the occurrence of it is 5.0% wider than the average width of the object in the top half of the image. After running 'testElephant.py' in the 'TestCode' folder, the output csv generated by this function indicates that the function correctly determines elephant's foot for images that fit the previous description.

Conclusion

The quality of a 3D printed object can be determined visually, these can also be determined through computer vision techniques and ingenuity. By reducing an image to binary black and white, formerly complex operations become as simple as counting pixels of a certain color. Despite the limited nature in the input of these functions, they serve as a proof of concept, that quality can be easily determined. With these functions, users will quickly and easily be able to estimate the quality of their work and have guidance to do better. Plans to expand upon these functions are in place, they will be placed on a Raspberry Pi with a camera module to allow them to be used while the print is being printed, this will reduce plastic waste from low quality prints, additional functions to determine PID Loop quality, nozzle cleanliness, etc. as well as allowing for increases in accuracy and less restrictions on image content.

References

[1] A. Rosebrock, “Detecting multiple bright spots in an image with python and opencv,” *PyImageSearch*, 17-Apr-2021. [Online]. Available: https://www.pyimagesearch.com/2016/10/31/detecting-multiple-bright-spots-in-an-image-with-python-and-opencv/. [Accessed: 07-Dec-2021].

[2] “Image thresholding,” *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html. [Accessed: 07-Dec-2021].

[3] A. G., “How to divide the image into 4 parts using opencv,” *Medium*, 14-Dec-2020. [Online]. Available: https://oleksandrg.medium.com/how-to-divide-the-image-into-4-parts-using-opencv-c0afb5cab10c. [Accessed: 07-Dec-2021].