






DETERMINING  
SHAPE AND  
SIZE OF AN  
OBJECT IN A  
GIVEN IMAGE

SUBMITTED BY:

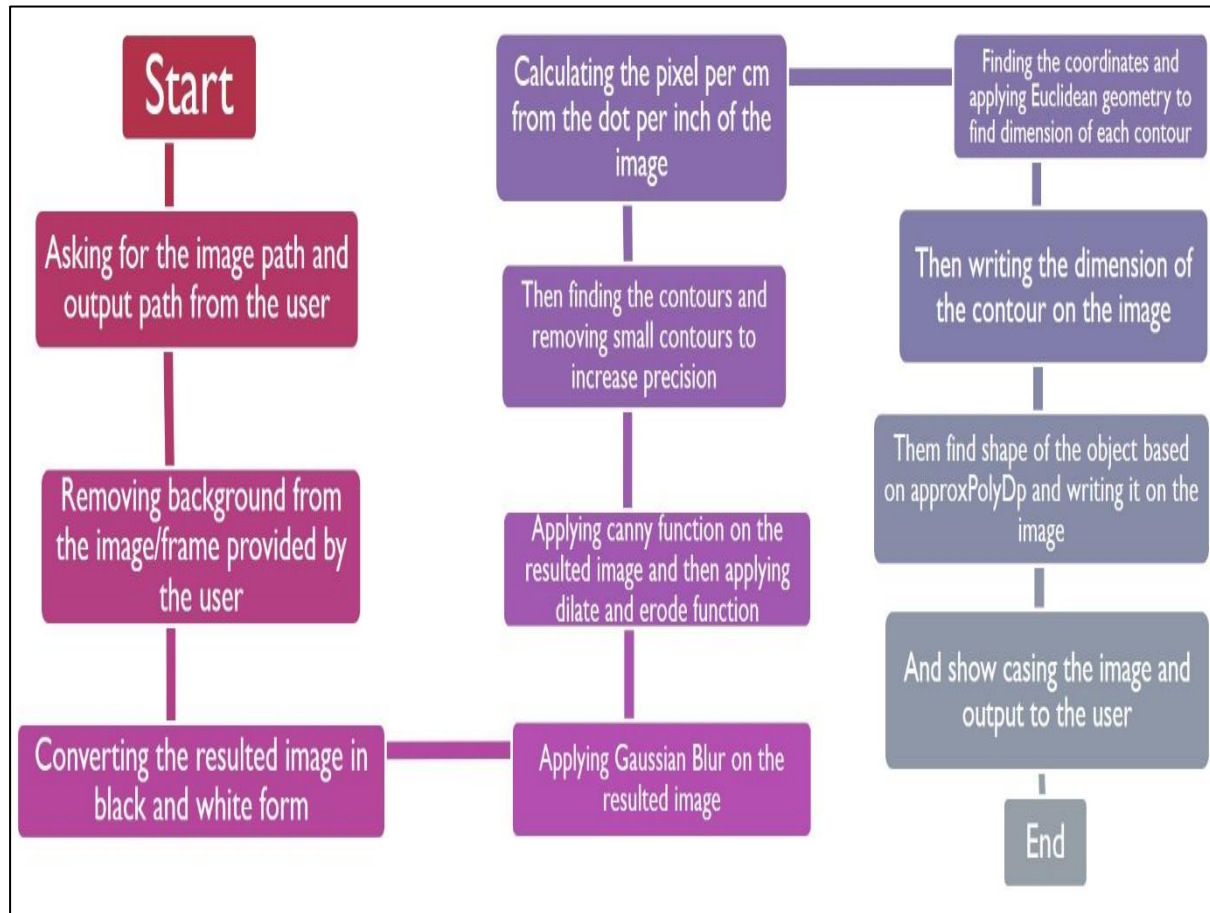
Serial Number	Name	Roll Number	Photo
1.	Jap Purohit	AU1940109	
2.	Nihar Patel	AU1940117	
3.	Mohit Prajapati	AU1940171	
4.	Raj Gariwala	AU1940118	
5.	Purvam Sheth	AU1940151	

## **Synopsis**

For certain applications where computer vision is needed instead of making physical observations, measuring objects dimension within an image or frame may be an essential capability. A simple step-by-step algorithm to isolate a target object and calculate its dimension will be covered in this implementation note. Along with determining the dimension of an object in a given image this implementation will also help in determining the shape of that particular object in a given image. This implementation is only applicable to 2D images.

## Techniques used, Flow charts

Overall flow chart of the program can be represent by the flow chart shown below:



### ▪ Concept of Canny Edge Detection:

Canny Edge Detection is an algorithm used to extract edges from images, and since it looks quite straightforward, I believe we can start with it.

The algorithm has four stages:

- **First:** Performs noise reduction with a Gaussian Blur;
- **Second:** Gets the gradient direction and magnitude with a Sobel kernel.
- **Third:** Applies non-maximum suppression, which removes unwanted pixels that are not part of a contour;

- **Fourth:** Applies the Hysteresis Thresholding that uses min and max values to filter the contours by the intensity gradient.

- **Concept of Dots Per Inch:**

DPI stands for "Dots Per Inch." DPI is used to measure the resolution of an image both on screen and in print. As the name suggests, the DPI measures how many dots fit into a linear inch. Therefore, the higher the DPI, the more detail can be shown in an image. It should be noted that DPI is **not** dots per square inch. Since a 600-dpi printer can print 600 dots both horizontally and vertically per inch, it actually prints 360,000 (600 x 600) dots per square inch. Also, since most monitors have a native resolution of 72 or 96 pixels per inch, they cannot display a 300-dpi image in actual size. Instead, when viewed at 100%, the image will look much larger than the print version because the pixels on the screen take up more space than the dots on the paper.

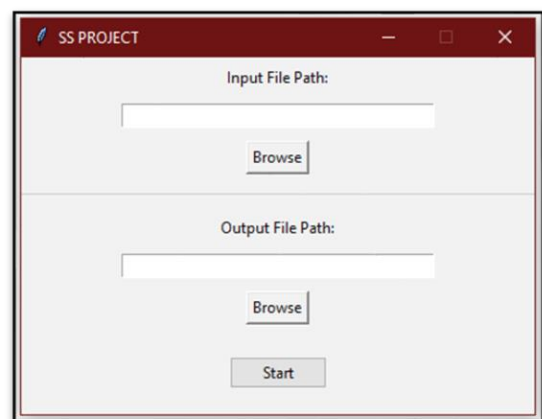
## **Description of user interface**

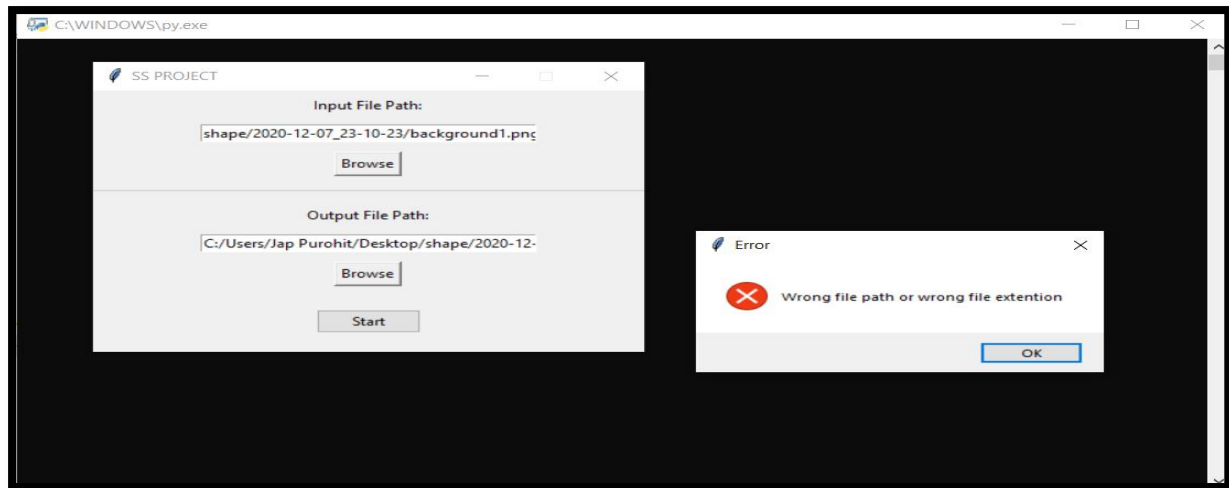
- User interface screen of the program is shown below:

Two paths are required to continue the program.

1. Input File Path:
2. Output File Path:

Input File Path only accepts .png and .jpg file formats. Another feature of this interface is that it has feature of validation i.e., it checks whether the image provided is in the proper format (.jpg or .png) if not then it will give error. And interface also has validation factor which check whether output path exist or not. If it does not exist then it shows error message. And interface again ask to the user for correct input.





- **Input file having depth of field**

This is one of the shots taken by iPhone X without any equipment having file format jpg. This file has a size of 3.64Mb having 96 dpi.

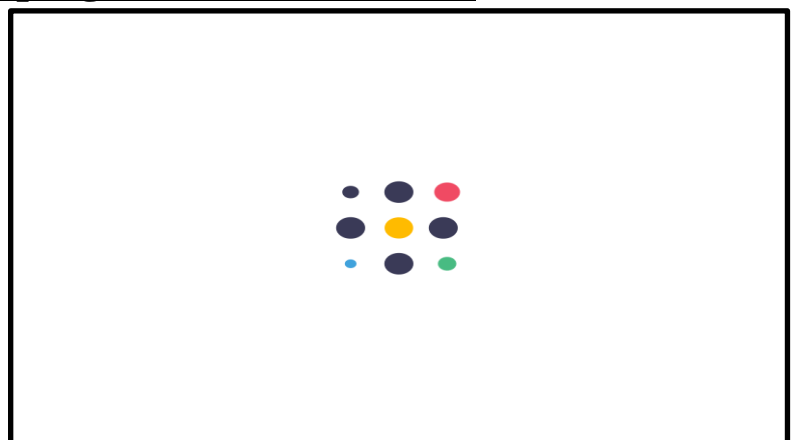
Image should be taken under proper lighting conditions and the subject should be sufficiently exposed to the light and photo should be taken perpendicular to the surface which we willing to find shape and dimension.

Photo should be taken such that its shadow is minimised.



- **Loading screen of the program is shown below:**

This loading screen will appear till process of measuring and detection of dimensions and shape respectively is run in the background for the given image. And it will automatically close the loading window when all



images will start to appear on the screen. This logic is implemented using concept of multithreading.

# Programs

Library used in making this project:

- scipy for measuring Euclidean distance
- imutils for contours
- remove\_bg\_api for removing background
- numpy for handling image data
- cv2 for image processing
- tkinter for Interface creation
- os library for handling with file system and validation of user input.

The code snippets are as follows:

1. This code snippet is about the user interface. The user interface is divided into two parts, top frame and bottom frame. The code also includes the concept of the multithreading because of the loading screen. So, while loading screen is appeared in back side the main function or image processing algorithm is implemented. Once it gets over loading screen is close and output is shown to the user.

```
top_frame = tk.Frame(master)
bottom_frame = tk.Frame(master)
line = tk.Frame(master, height=1, width=400, bg="grey80", relief='groove')
filePathDict = {}

input_path = tk.Label(top_frame, text="Input File Path:")
input_entry = tk.Entry(top_frame, width=40)
browse1 = tk.Button(top_frame, text="Browse", command = inputFunction)

output_path = tk.Label(bottom_frame, text="Output File Path:")
output_entry = tk.Entry(bottom_frame, text="", width=40)
browse2 = tk.Button(bottom_frame, text="Browse", command= outputFunction)

begin_button = ttk.Button(bottom_frame, text='Start', command= lambda: helloCallBack(filePathDict))

top_frame.pack(side=tk.TOP)
line.pack(pady=10)
bottom_frame.pack(side=tk.BOTTOM)

input_path.pack(pady=5)
input_entry.pack(pady=5)
browse1.pack(pady=5)

output_path.pack(pady=5)
output_entry.pack(pady=5)
browse2.pack(pady=5)

begin_button.pack(pady=20)
master.title("SS PROJECT")
master.resizable(0,0)
master.mainloop()
```

2. This code snippet is the logic for the background removing process and making new folder to the path provided by the user for storing all output in this new folder with folder name as *timestamp* when this

```
print("in main function")
print("background removing process")
ct = datetime.datetime.now
output_path_folder = os.path.join(filePathDict['outputPath'], datetime.datetime.now().strftime('%Y-%m-%d_%H-%M-%S'))
os.mkdir(output_path_folder)

output_path = output_path_folder + '/background.png'
removebg = RemoveBg('9x7YHu3oqrjyNAADoTcCL58c')
print("background removing process")
# Send and save the finished image
image = removebg.remove_bg_file(input_path= filePathDict['inputPath'], out_path= output_path, size="preview", raw=False)

# Print path
print("Image was saved along the path: {}".format(image))
os.chdir(output_path_folder)
```

application is started to work.

- This code snippet includes the logic of image processing and how edge detection is done using canny but before that original image is converted into gray image then it is converted into blur and then apply canny, then dilate and erode the image for increasing the accuracy of the application. Then finding the contours and removing unnecessary contours. Then finally draw contours and for measurement of the object we have used the concept of DPI (Dots Per inch) and PPI (Pixel Per Inch). Then finally labeling the shape and dimension on the image.

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (9, 9), 0)
edged1 = cv2.Canny(blur, 50, 100)
edged2 = cv2.dilate(edged1, None, iterations=1)
edged3 = cv2.erode(edged2, None, iterations=1)

# Find contours
cnts = cv2.findContours(edged3.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)

# Sort contours from left to right as leftmost contour is reference object
(cnts, _) = contours.sort_contours(cnts)

# Remove contours which are not large enough
cnts = [x for x in cnts if cv2.contourArea(x) > 850]

pixel_per_cm = pixel_per_cm_function(im1.info['dpi'][0])
font = cv2.FONT_HERSHEY_SIMPLEX

# Draw remaining contours
for cnt in cnts:
    box = cv2.minAreaRect(cnt)
    box = cv2.boxPoints(box)
    box = np.array(box, dtype="int")
    box = perspective.order_points(box)
    (tl, tr, br, bl) = box
    x, y, w, h = cv2.boundingRect(cnt)
    cv2.drawContours(originalImage, [box.astype("int")], -1, (0, 0, 255), 2)
    mid_pt_horizontal = (tl[0] + int(abs(tr[0] - tl[0])/2), tr[1] + int(abs(tr[1] - tl[1])/2))
    mid_pt_verticle = (tr[0] + int(abs(tr[0] - br[0])/2), tr[1] + int(abs(tr[1] - br[1])/2))

    wid = (euclidean(tl, tr) / pixel_per_cm) * 1.09
    ht = (euclidean(tr, br) / pixel_per_cm) * 1.09
    approx = cv2.approxPolyDP(cnt, 0.009 * cv2.arcLength(cnt, True), True)
    if (len(approx) <= 10):
        cv2.putText(originalImage, "%0.2f cm"%(wid), (int(mid_pt_horizontal[0] - 15), int(mid_pt_horizontal[1] - 10)), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 2)
        cv2.putText(originalImage, "%0.2f cm"%(ht), (int(mid_pt_verticle[0] + 10), int(mid_pt_verticle[1])), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 2)

    if (len(approx) == 4):
        if w == h:
            #cv2.putText(image, "%0.2f cm"%(wid), (int(mid_pt_horizontal[0] - 15), int(mid_pt_horizontal[1] - 10)), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 2)
            cv2.putText(originalImage, "Square", (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 2)
        else:
            cv2.putText(originalImage, "Rectangle", (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 2)
```

- This final code snippet is for displaying result to the user and saving the result obtained at each step of the image processing to the output file provided by the user.

```
def show_images(images):
    # Function to show array of images (intermediate results)
    for i, img in enumerate(images):
        half = cv2.resize(img, (897, 539), fx = 0.1, fy = 0.1)
        cv2.imshow("image_" + str(i), img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    # Initialize api wrapper


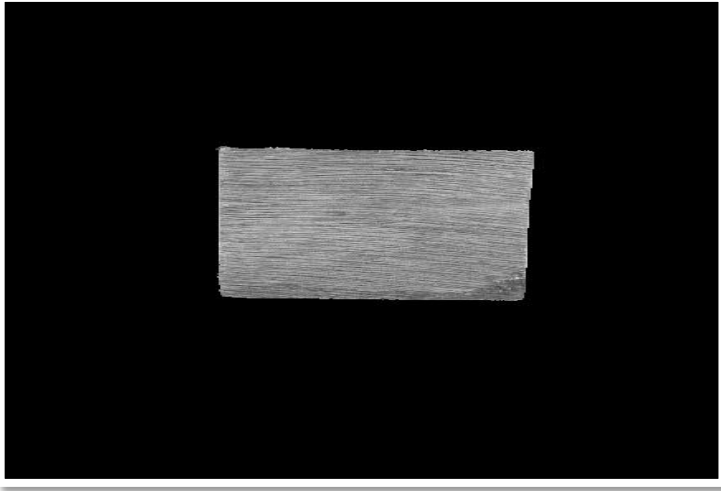
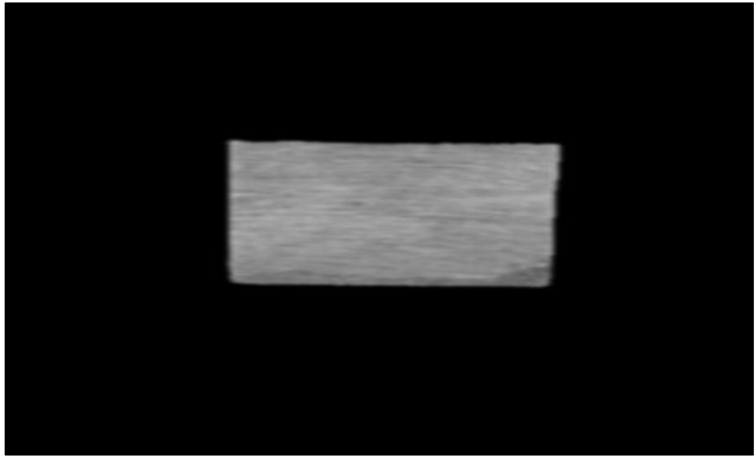
def save_images(images):
    image_dict = Convert(images)
    for key in image_dict:
        filename = key+'.jpg'
        cv2.imwrite(filename, image_dict[key])
```

Lastly the code is saved in the format of `‘.py’`. And this whole code is implemented in the *“python 3.7.3”*.



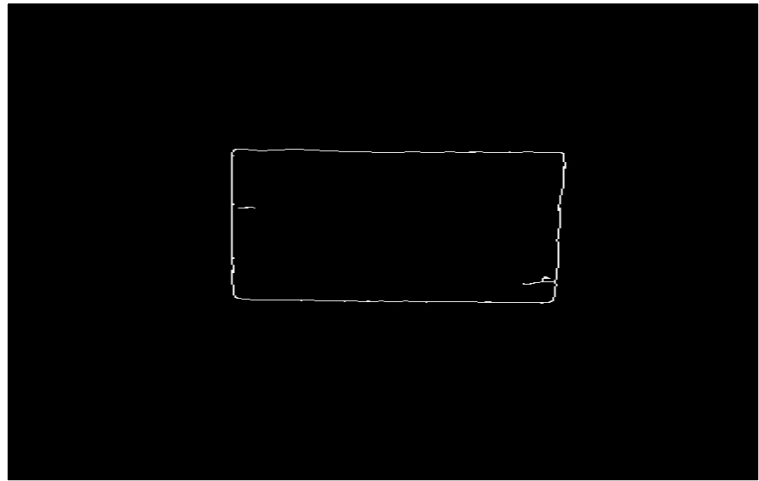
## Results

While displaying the result to user. The application will display following images.

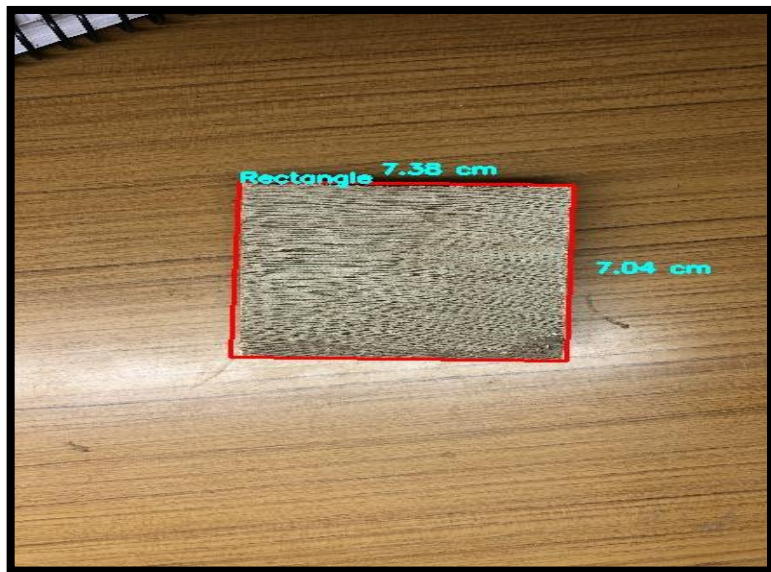
<u>Image Number and Description</u>	<u>Image</u>
<b><u>Image 1:</u></b> First image is which is showed to user is background removed image. This image is saved as background image.	
<b><u>Image 2:</u></b> Second image is grey (black and white) image which converted from the background removed image. This is saved as gray image.	
<b><u>Image 3:</u></b> After that grey image is made blur to get ease in edge recognition. This image is saved as blur image.	



**Image 4:** After blur image edge detection image is shown to the user after applying canny, dilate and erode function of openCV2 library. This image is saved as edged3.



**Image 5:** Final image which is shown to the user is the image with drawn borders of detected edges and determining the shape and dimension of the image. This image is saved as originalImage.



So only 5 main images are displayed to the user. All images which were obtained after each function performed related image processing is saved in the new folder with folder name as *time\_stamp* when this application is started. The images saved are background, blur, edged1, edged2, edged3, gray, image, originalImage and all these files in jpg format except background image. Below image is example show it is stored.



## Conclusion

On a closing note, this project was a great chance for us to learn the principles of Canny i.e., Gaussian convolution to smooth the input image and remove noise. In addition to the fact that we learned substantially more about it and applied into the project. In addition to this, there are many other elements for resizing of an image by decimation and interpolation. The big task while making this project for us was to develop a user interface which would take the input and output location of the image into which validation for the image extension is also applied, since this was the first time, we were supposed to merge the locations into a same interface. But apart from that, engaging with the team was a remarkable opportunity, and getting their input on how to go forward to tackle a specific challenge and ultimately completed the project.

Each member of the group made an equal contribution to the analysis of the project. With Jap and Purvam being the better coder implemented most of the code. The interface which takes the location of the image file was made by Mohit and Raj which was then implemented into the code. Nihar contributed in making of the report and it's formatting. And last but not the least the presentation making was done by Raj. Overall, the report was a collaborative activity by the whole group, and because of the group work, it was possible.

## Sources:

*Calculator to compute the Distance or Size of an Object in a photo Image.* (n.d.). Scantips.  
<https://www.scantips.com/lights/subjectdistance.html>

K. (n.d.). *Tools & API* –. Remove.Bg. <https://www.remove.bg/tools-api>

*multiprocessing — Process-based parallelism — Python 3.9.1 documentation.* (n.d.).  
Python.Org. <https://docs.python.org/3/library/multiprocessing.html>

OpenCV. (2020, October 13). *openCv*. <https://opencv.org/>

*OpenCV: Contours : Getting Started.* (n.d.). OpenCv.  
[https://docs.opencv.org/master/d4/d73/tutorial\\_py\\_contours\\_begin.html](https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html)

*Python - Multithreaded Programming - Tutorialspoint.* (n.d.). Tutorialspoint.  
[https://www.tutorialspoint.com/python/python\\_multithreading.htm](https://www.tutorialspoint.com/python/python_multithreading.htm)

*REAL TIME OBJECT MEASUREMENT / OpenCV Python (2020).* (2020, June 14). [Video].  
YouTube. [https://www.youtube.com/watch?v=tk9war7\\_y0Q](https://www.youtube.com/watch?v=tk9war7_y0Q)

*tkinter — Python interface to Tcl/Tk — Python 3.9.1 documentation.* (n.d.). Python.Org.  
<https://docs.python.org/3/library/tkinter.html>

Wesolowski, M. (2014, November). *Using MATLAB to Measure the Diameter of an Object within an Image.*  
[https://www.egr.msu.edu/classes/ece480/capstone/fall14/group05/docs/app\\_matt.pdf](https://www.egr.msu.edu/classes/ece480/capstone/fall14/group05/docs/app_matt.pdf)

*What is the difference between Dots Per Inch (DPI) and Pixels Per Inch (PPI)? | Sony USA.*  
(2013, April 19). Sony.Com.  
<https://www.sony.com/electronics/support/articles/00027623>