Quality Control

For the image processing and recognition, we used computer vision module of python, called OPENCV. The entire code is divided into 4 main parts namely:

1. Web-cam setup
2. Image processing
3. Object dimensions and calculations
4. Python to Arduino Cloud communication

1.Web – cam setup:

For accessing the live feed, droid-cam client (android) and IP address of Wi-Fi was used. The while loop executes and displays the live feed to grab and return the frame. If frame is not returned, it will break out of the loop. If camera is working properly, we can capture the frame by pressing space-bar key, which will also save our image locally. The image counter increments by the times the spacebar was pressed, and all the clicked images are saved in a separate directory. A for loop iterates over all these images and depending on the variance threshold (120) obtained by Laplacian attribute, the clear image is selected for image processing.

2. Image Processing:

The last clicked frame is used for image processing. It is pre-processed by converting it grayscale and blurring to reduce distortions and noise. After that, image thresholding, mainly THRESH_BINARY and THRESH_BINARY_INV was applied. For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise it is set to a maximum value. This threshold is THRESH_BINARY and for THRESH_BINARY_INV it's the exact opposite. This was done to differentiate between bigger outer and smaller inner contours(holes). To reduce noise even further morphological transformation and kernel is used. Kernel is a structuring element, in this case for proper circular contour identification, elliptical shaped (MORPH_ELLIPSE) kernel is used.

To get the contours, findContours function was used with the first argument being the morphed image. As the smaller holes are nested within the bigger circle, RETR_LIST (2nd argument) was used to simply store all the contours without creating any hierarchical relationship. CHAIN_APPROX_SIMPLE (3rd argument) considers only the endpoints. A for loop was executed to get x -y coordinates and areas of contours and were stored in separate lists. To visualize the contours, drawContours and circle functions were used to mark out the circular contours and its centre.

3. Object dimensions and calculations:

Area and x-y coordinates were zipped(zip) and sorted is descending order to make the outer contour the first element. Considering the memory complexity, applied list comprehension to filter out the first element without taking up any extra memory. Separate lists were created to store only the smaller circles x-y coordinates. A user-defined function was implemented to return the centre coordinates of an imaginary line joining two/three holes depending on the operation performed on the object. Coordinates of outer circle and coordinates of midpoint between small holes were used to find the Euclidean distance between them. Another user-defined function, reject_status which takes the distance as an argument returns a reject variable. It returns 0 if distance is less than 10% of outer circle radius (object is accepted), else 1 (object is rejected).

4.Python to Arduino Cloud communication:

The code to connect a python script to Arduino cloud dashboard was taken directly from the official Arduino Cloud Client Documentation. For its proper functionality, only the Device ID and secret key of ESP8266 were needed. The rejected variable in the dashboard increments depending on the value of reject variable which is sent to the cloud client through this python script.

Dependencies/modules used in this code:

1. cv2
2. time
3. logging
4. nest_asyncio
5. math
6. sys
7. ArduinoCloudClient from arduino_iot_cloud
8. import glob
9. from pathlib import Path