****

**EMT Assignment 1 Report**

Computer Vision System

Lim Song Hao, Iain

PS01

S10184549A

**1. Introduction**

Computer vision is a powerful tool in connecting computers to the physical world. It enables various task to be automated, fast and consistent. This can include identifying defects in production lines, facial recognition in security systems and even the for the rovers on mars to navigate the foreign terrain. With computer vision becoming more relevant to modern technologies, it is important to understand how it works.

This project aims to create a computer vision program for detection and sorting of Lego bricks. Visual Studio Code would be used as the code editor for this project with HTML used for GUI and Python used for the main code.

**2. Graphic User control (GUI)**

The GUI consist of a level selection for the video, a video display, start and stop buttons, options and a table for the blocks detected.

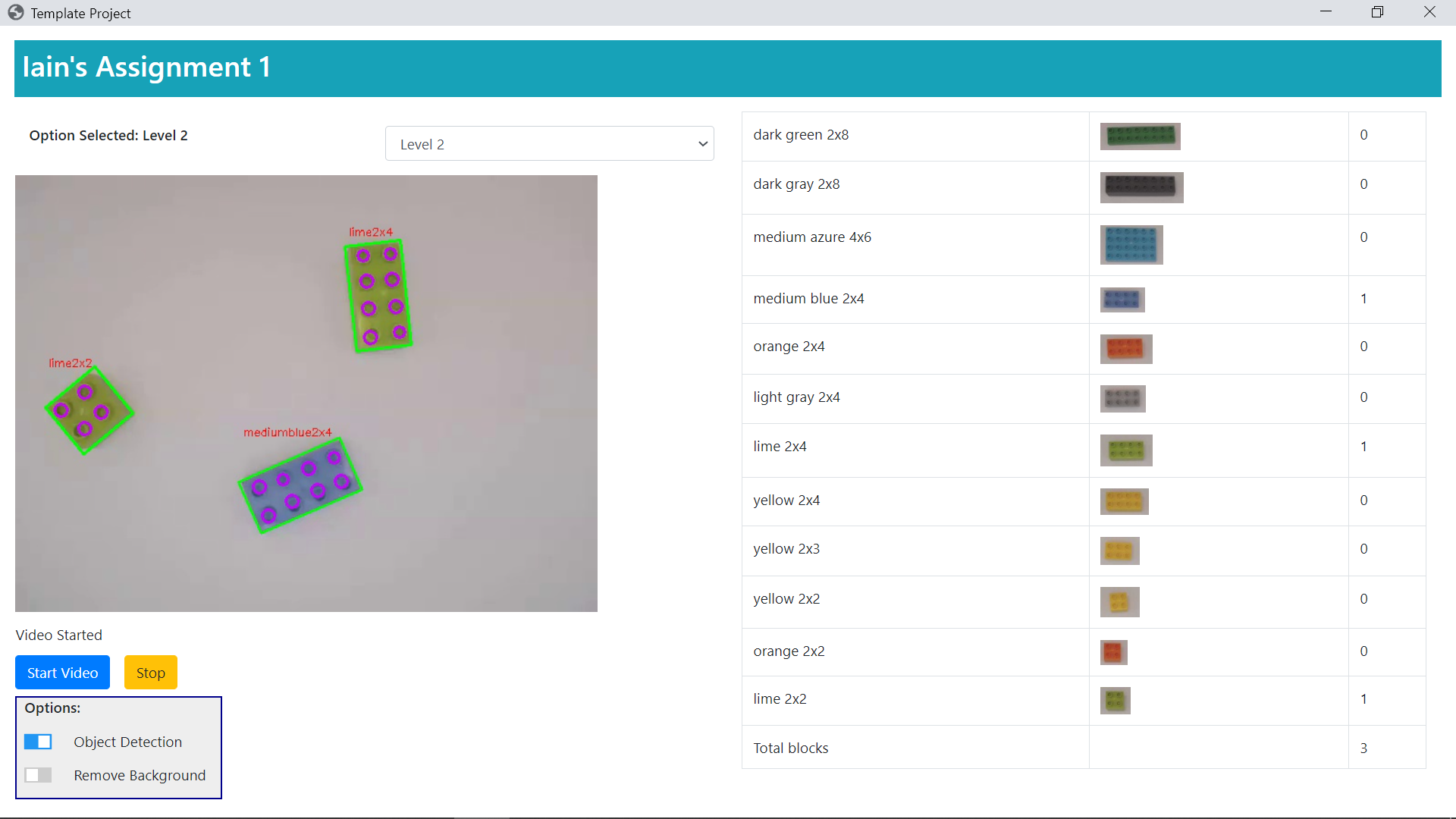


Figure 1: The HTML GUI of the Lego block detection program

Video level selection

* User can choose between level 1 to 3 which are set in the main.py code.

Video display

* The output video will depend in the selected level and the options selected.

Start and Stop button

* A new video can be initiated only when the current video has finished or stopped.

Options

* This small menu allows user to toggle the different settings for the output video.
* Toggle object detection to show blocks contour, location, circles, and count.
* Toggle Remove background to show the mask for all the Lego blocks

Table of blocks detected

* Displays the number of each block type currently on the video display

For the toggle switch, HTML style code is adapted from [1]

**3. Computer Vision**

The Lego block detection using computer vision in this code mainly comprises of two main sections, masking of the individual colors and detection of block type. The approach taken for Lego block detection and sorting is:

1. Creating a colored mask for each color
2. Find the contour of the colored mask for bounding box
3. Detect the number circles in the bounding box
4. Draw circles and minimum area rectangle
5. Determine the block based on color mask used and number of circles

hsv\_frame = cv2.cvtColor(newframe, cv2.COLOR\_BGR2HSV)

low\_azure = np.array([180\*188/360, 255\*0.15, 255\*0.38])

high\_azure = np.array([180\*212/360, 255\*0.75, 255\*0.78])

azure\_mask = cv2.inRange(hsv\_frame, low\_azure, high\_azure)

azure\_mask = cv2.morphologyEx(azure\_mask, cv2.MORPH\_CLOSE, kernel)

azure = cv2.bitwise\_and(newframe, newframe, *mask*=azure\_mask)

.

.

.

To create a mask for individual colors, a Hue Saturation Value (HSV) version of the current frame is created.

A mask of each block colors is then created based on the lowest and highest threshold of the HSV value of the block colors. The values for the blocks HSV is determined using GIMP, a free open source graphic editor for image manipulation and editing.

After getting the mask for the block color, closing is performed to remove any foreground noise from the block. The mask is then applied to the current video frame to create another image showing only the specific block color.

This process is then repeated for each block color except for light and dark gray where they are masked together. This code is adapted from [2] and the process can be seen in Figure 2 below.

Figure 2: This image shows the color masking process for medium blue blocks from left to right.

Original frame (Far Left). Binary masking (Left). Closing (Right). Colored mask (Far Right)

After masking of the current frame, the circle detection and contour detection are performed on each coloured mask. This is used to determine the locations and different types of blocks in the frame.

cframe = cv2.cvtColor(lime,cv2.COLOR\_BGR2GRAY)

cframe = cv2.GaussianBlur(cframe,(3,3),0)

circles = cv2.HoughCircles(cframe,cv2.HOUGH\_GRADIENT,1,24,*param1*=70,*param2*=9,*minRadius*=3,*maxRadius*=8)

limegrayed = cv2.cvtColor(lime, cv2.COLOR\_BGR2GRAY)

contours,hierarchy=cv2.findContours(limegrayed,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)

*for* pic, contour *in* enumerate(contours):

area = cv2.contourArea(contour)

*if*(area>2500):

   countc = 0

    x,y,w,h = cv2.boundingRect(contour)

*if* circles is not None:

     circles = np.uint16(np.around(circles))

*for* i *in* circles[0,:]:

*if*(x<i[0] and i[0]<(x+w) and y<i[1] and i[1]<(y+h)):

          cv2.circle(newframe,(i[0],i[1]),i[2],(255,0,200),2)

         countc=countc+1

rect = cv2.minAreaRect(contour)

    box = cv2.boxPoints(rect)

    box = np.int0(box)

    newframe = cv2.drawContours(newframe,[box],0,(0,255,0),2)

*if* countc > 4:

     countlime2x4 += 1

     cv2.putText(newframe,"lime2x4",(x,y-8),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))

*else*:

     countlime2x2 += 1

      cv2.putText(newframe,"lime2x2",(x,y-8),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))

For circle detection, colored frame is first converted to a grayscale image. The grayscale image is then blurred using GuassianBlur(), except for yellow blocks since the circle detection would not be able to detect the circle edge well for yellow. The HoughCirles() function is then used to extract the locations and size of the circles. The main parameters modified in the code is the 4th parameter for the distance needed between the circles, param1 for the canny edge detection in the function, param2 for the threshold for circles detection and the last two parameters for the minimum and maximum size of the circles. By modifying each of these parameters for each block colors, the detection of the circle in the colored mask is very consistent.

To determine the contour of the block, findContours() function is applied to the grayscale color mask which obtains all the blocks detected in the color mask. The block identification code then go through all the blocks detected.

First, all the blocks that are too small are rejected. Next, a bounding rectangle is created to count all the circles inside of it. A minimum area rectangle is then created to outline the parameter of the block. The type of size of the Lego block is determined by the number of circles in the bounding rectangle. For this color lime, if circles is more than 4, the block must be a 2x4, else is can only be a 2x2. The count for the block is then incremented.

The block identification code is repeated for all the blocks detected.

The code shown above is only for the lime color mask, the others blocks colors follow a similar approach with the main difference being the HoughCirles() parameter values and if the blocks need a if statement to determine the type of blocks since some colors have only one size.

The code for Hough circle detection is adapted from [3] and the code for bounding box and minimum area rectangle is adapted from [4].

img\_send\_to\_js(newframe, "output")

text\_send\_to\_js(countgreen2x8, "green2x8count")

.

.

.

text\_send\_to\_js(countyellow2x4, "yellow2x4count")

total\_blocks = countgreen2x8 + countdarkgray2x8 + countlightgray2x4 + countlime2x2 + countlime2x4 + countmediumazure4x6 + countmediumblue2x4 + countorange2x2 + countorange2x4 + countyellow2x2 + countyellow2x3 + countyellow2x4

text\_send\_to\_js(total\_blocks, "totalcount")

After identifying all the blocks, the new frame with all the boxes, circles and block labels is sent to the output image in the HTLM. The count for each type of Lego blocks are also sent to the HTML with the total block count.

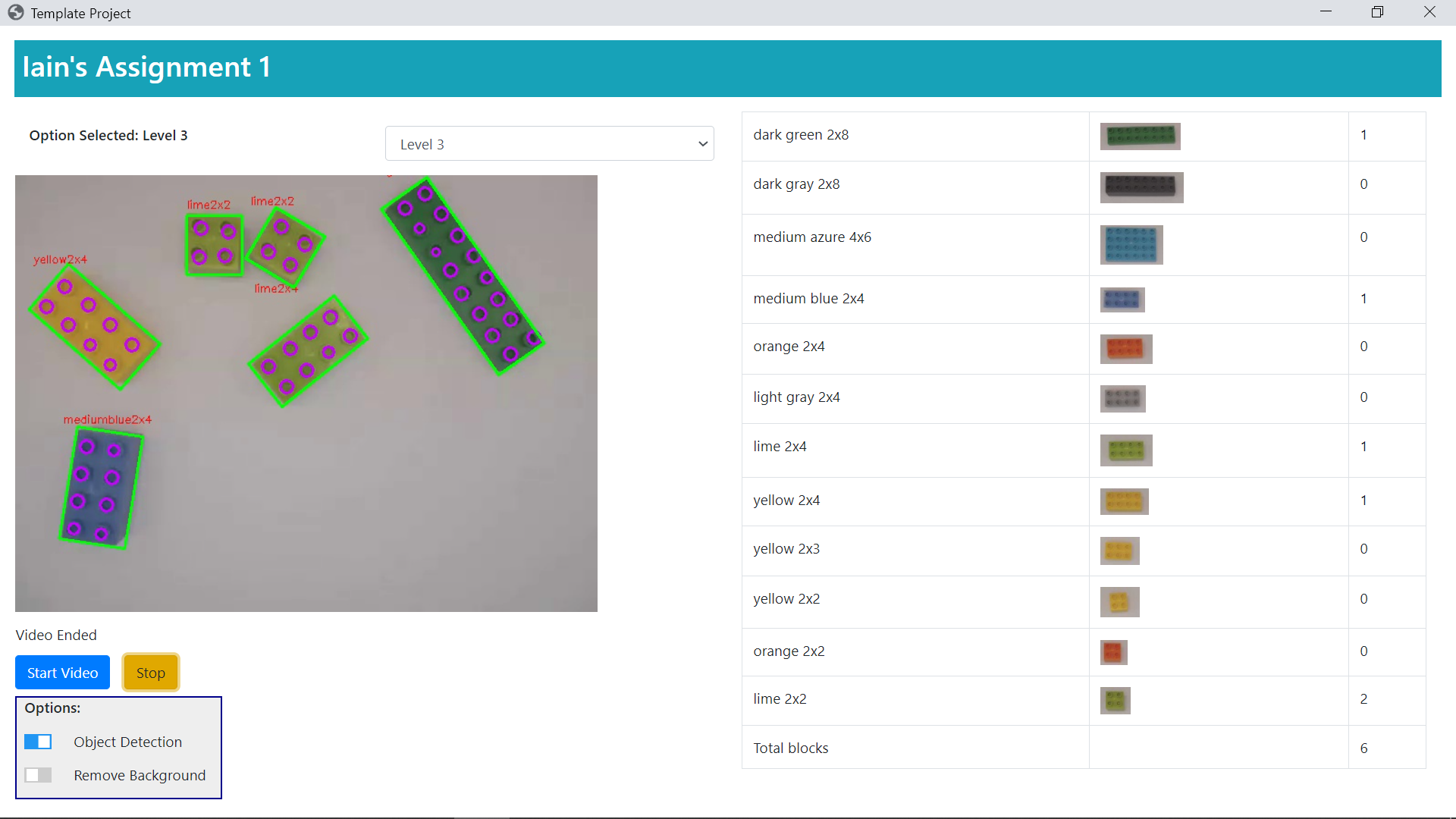


Figure 3: This image shows the output image after the object detection code with each Lego block count on the right.

The options to control the video output is controlled by toggle switches these switches which are actually check boxes with a different style. To receive the input, an “onclick” event for each toggle switch is used.

@eel.expose

def toggle\_detect():

  global detect

  detect = not detect

@eel.expose

def toggle\_background():

  global hide\_background

  hide\_background = not hide\_background

These two functions help toggle the global variable detect and hide\_background. When detect is false, the detection code for blocks and circles are skipped. When hide background is true, the different mask for the different colors are combined and applied to the current frame which the boxes, circles and block labels would added on top.

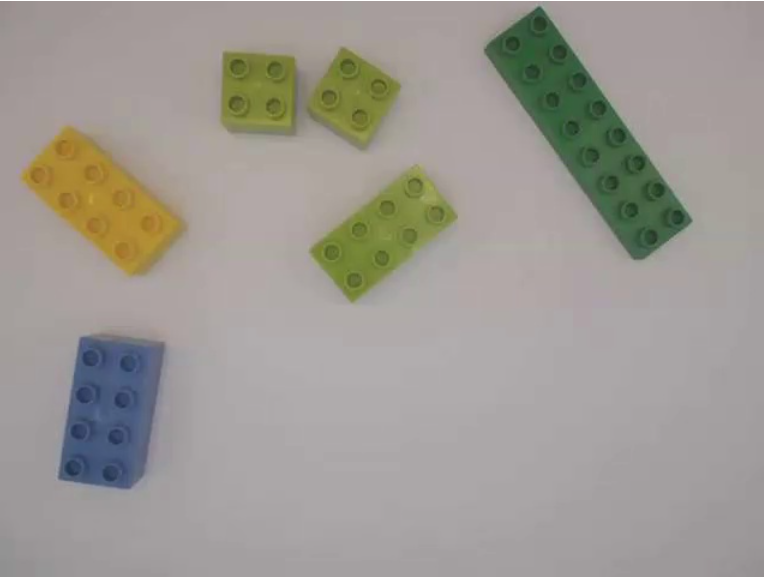
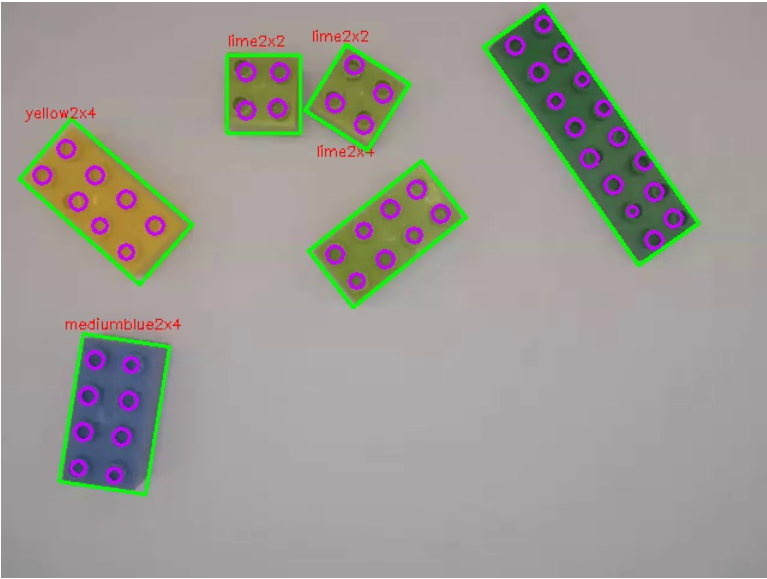
 

Figure 4: No detection with background Figure 5: Detection with background

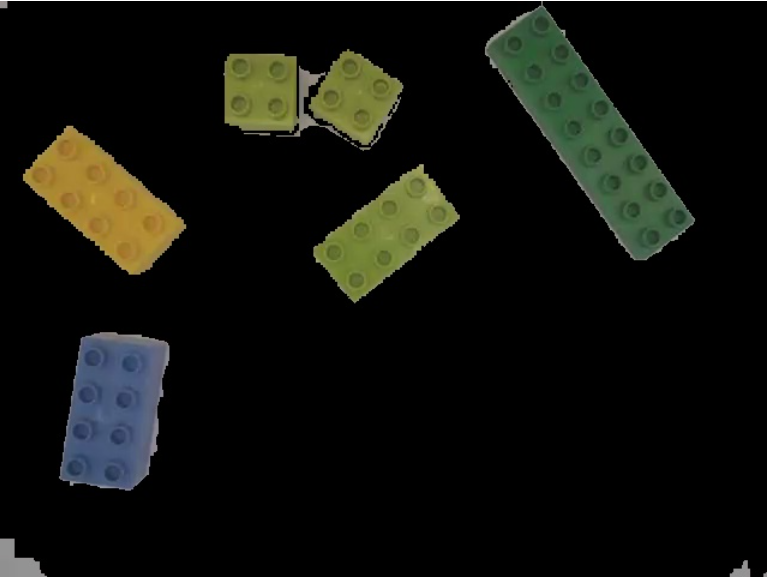
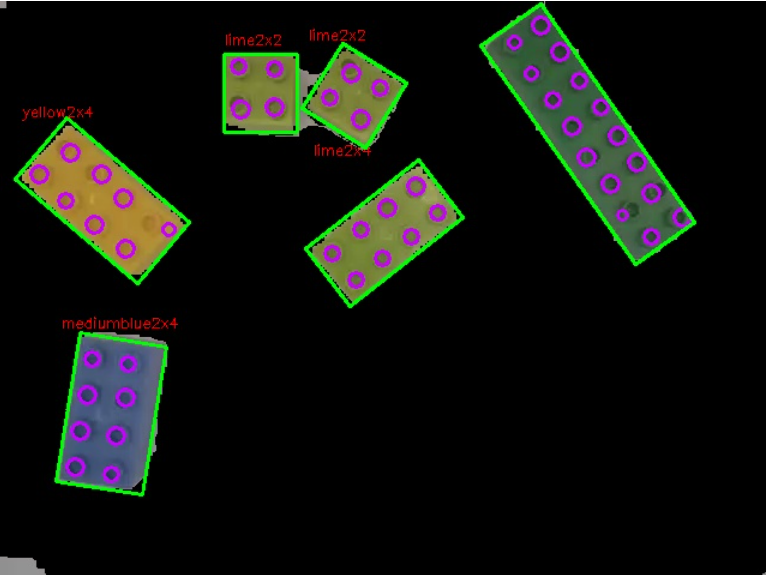
 

Figure 6: No detection without background Figure 7: Detection without background

**4. Constraints**

One of the constraints of this program is that in one or two videos where the circle in the center of the block is detected occasionally. This is only an issue for 2x4 yellow blocks since it is the only color with 2x4 and 2x3.

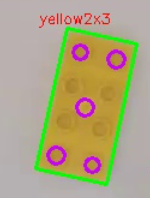
  

Figure 8: This image shows the issues of detecting the center circle.

Image of the block without circle detection (Left). Image of normal circle detection (Middle). Image of failed circle detection (Right).

**5. Results and Discussion**

Other than detecting the center circle, most of the Lego video clips gives the correct results consistently. With some circles being missing occasionally which does not affect the type of Lego block identification.

The finished code was much better than I expected. Although most of the time was spent on fine tuning the mask HSV values and the Hough circle parameters, there were some ways I worked around the extra fine tuning required using concepts from previous lessons and practical sessions. This includes using closing on the mask to reduce the need for finding the extreme low and high HSV values which could overlap with other colors and Gaussian blur before Hough circle detection to reduce the noise affecting the detection.

**6. Conclusion**

This project was successful in detecting the different Lego blocks for the various given videos. Through this unique and interesting computer vision project, I have gained a much better appreciation and understanding of the various image processing technics.

**7. Reference**

|  |  |
| --- | --- |
| [1] | W3Schools, "How TO - Toggle Switch," [Online]. Available: https://www.w3schools.com/howto/howto\_css\_switch.asp. [Accessed 2 January 2021]. |
| [2] | S. Canu, "pysource," 31 January 2018. [Online]. Available: https://pysource.com/2018/01/31/object-detection-using-hsv-color-space-opencv-3-4-with-python-3-tutorial-9/. [Accessed 1 January 2021]. |
| [3] | A. Rosebrock, "PyImageSearch," 21 July 2014. [Online]. Available: https://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/. [Accessed 2 January 2021]. |
| [4] | OpenCV.ORG, "Contour features," [Online]. Available: https://docs.opencv.org/3.4/dd/d49/tutorial\_py\_contour\_features.html. [Accessed 2 January 2021]. |

**8. Appendix**

index.HTML

<!doctype html>

<html lang="en">

<head>

    <title>Template Project</title>

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>

    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></script>

    <script type="text/javascript" src="/eel.js"></script>

    <script type="text/javascript" src="./index.js"></script>

    <style>

      .switch {

        position: relative;

        display: inline-block;

        width: 30px;

        height: 17px;

      }

      .switch input {

        opacity: 0;

        width: 0;

        height: 0;

      }

      .slider {

        position: absolute;

        cursor: pointer;

        top: 0;

        left: 0;

        right: 0;

        bottom: 0;

        background-color: #ccc;

        -webkit-transition: .4s;

        transition: .4s;

      }

      .slider:before {

        position: absolute;

        content: "";

        height: 13px;

        width: 13px;

        left: 2px;

        bottom: 2px;

        background-color: white;

        -webkit-transition: .4s;

        transition: .4s;

      }

      input:checked + .slider {

        background-color: #2196F3;

      }

      input:focus + .slider {

        box-shadow: 0 0 1px #2196F3;

      }

      input:checked + .slider:before {

        -webkit-transform: translateX(13px);

        -ms-transform: translateX(13px);

        transform: translateX(13px);

      }

*/\* Rounded sliders \*/*

      .slider.round {

        border-radius: 17px;

      }

      .slider.round:before {

        border-radius: 50%;

      }

      </style>

  <style>

    fieldset {

      background-color: #eeeeee;

      border: 2px solid darkblue;

    }

    input {

      margin: 5px;

    }

  </style>

</head>

*<!-- <body onload="py\_video()"></body> -->*

<body>

*<!-- HTML STARTS HERE -->*

*<!-- Type your codes here -->*

    <div class="container-sm|md|lg|xl p-2 m-3 bg-info">

        <h2 class="text-white">Iain's Assignment 1</h2>

    </div>

    <div class="container-sm|md|lg|xl px-3 mx-3">

        <div class="row md-6">

*<!-- video, left side -->*

            <div class="col-md-6">

                <div class="row my-3">

                    <div class="col-md-6">

                      <h6 class="m-0" id="p1"> Select the level </h6>

                    </div>

                    <div class="col-md-6">

                        <select class="form-control" id="idOption">

                          <option value=""> </option>

                          <option value="1"> Level 1</option>

                          <option value="2"> Level 2</option>

                          <option value="3"> Level 3</option>

                        </select>

                    </div>

                </div>

                <div class="row">

                  <img id="output" src="">

                </div>

                <div class="row my-1">

                  <p class="my-2" id="process"> No videos running </p>

                </div>

                <div class="row">

                  <button onclick="eel.video\_feed()" id="btn1" type="button" class="btn mr-3 btn-primary">Start Video</button>

                  <button onclick="eel.stop\_video\_feed()" id="btn2" type="button" class="btn mr-3 btn-warning">Stop</button>

                </div>

                <div class="row my-2">

                  <fieldset>

                    <h6 class="ml-2 mb-3">Options:</h6>

                    <div class="row mx-2 my-2">

                      <label class="switch mt-1">

                        <input type="checkbox" id="switch1" onclick="eel.toggle\_detect()">

                        <span class="slider"></span>

                      </label>

                      <h7 class="ml-4 mr-2">Object Detection</h7>

                    </div>

                    <div class="row mx-2 my-2">

                      <label class="switch mt-1">

                        <input type="checkbox" id="switch1" onclick="eel.toggle\_background()">

                        <span class="slider"></span>

                      </label>

                      <h7 class="ml-4 mr-2">Remove Background</h7>

                    </div>

                  </fieldset>

                </div>

            </div>

*<!-- table, right side -->*

            <div class="col-md-6">

                <table class="table table-bordered">

                    <tbody>

                      <tr>

                        <td>dark green 2x8</td>

                        <td><img id="green2x8img" src=""></td>

                        <td id="green2x8count">0</td>

                      </tr><tr>

                        <td>dark gray 2x8</td>

                        <td><img id="darkgray2x8img" src=""></td>

                        <td id="darkgray2x8count">0</td>

                      </tr><tr>

                        <td>medium azure 4x6</td>

                        <td><img id="mediumazure4x6img" src=""></td>

                        <td id="mediumazure4x6count">0</td>

                      </tr><tr>

                        <td>medium blue 2x4</td>

                        <td><img id="mediumblue2x4img" src=""></td>

                        <td id="mediumblue2x4count">0</td>

                      </tr><tr>

                        <td>orange 2x4</td>

                        <td><img id="orange2x4img" src=""></td>

                        <td id="orange2x4count">0</td>

                      </tr><tr>

                        <td>light gray 2x4</td>

                        <td><img id="lightgray2x4img" src=""></td>

                        <td id="lightgray2x4count">0</td>

                      </tr><tr>

                        <td>lime 2x4</td>

                        <td><img id="lime2x4img" src=""></td>

                        <td id="lime2x4count">0</td>

                      </tr><tr>

                        <td>yellow 2x4</td>

                        <td><img id="yellow2x4img" src=""></td>

                        <td id="yellow2x4count">0</td>

                      </tr><tr>

                        <td>yellow 2x3</td>

                        <td><img id="yellow2x3img" src=""></td>

                        <td id="yellow2x3count">0</td>

                      </tr><tr>

                        <td>yellow 2x2</td>

                        <td><img id="yellow2x2img" src=""></td>

                        <td id="yellow2x2count">0</td>

                      </tr><tr>

                        <td>orange 2x2</td>

                        <td><img id="orange2x2img" src=""></td>

                        <td id="orange2x2count">0</td>

                      </tr><tr>

                        <td>lime 2x2</td>

                        <td><img id="lime2x2img" src=""></td>

                        <td id="lime2x2count">0</td>

                      </tr>

                      </tr><tr>

                        <td>Total blocks</td>

                        <td></td>

                        <td id="totalcount">0</td>

                      </tr>

                    </tbody>

                  </table>

            </div>

        </div>

    </div>

*<!-- HTML ENDS HERE -->*

</body>

</html>

index.js:

eel.setup()

eel.expose(updateImageSrc)

function updateImageSrc(*val*, *id*) {

  let elem = document.getElementById(id);

  if (val == "")

  {

    elem.src = "";

  }

  else

  {

    elem.src = "data:image/jpeg;base64," + *val*;

  }

}

eel.expose(updateTextSrc)

function updateTextSrc(*val*,*id*) {

  document.getElementById(*id*).innerHTML = *val*;

}

eel.expose(updateImageSrc)

function updateImageSrc(*val*, *id*) {

  let elem = document.getElementById(*id*);

  elem.src = "data:image/jpeg;base64," + *val*;

}

function py\_video() {

   eel.video\_feed()()

}

let captureActive = true;

$(window).keypress(function(*e*) {

  if (*e*.key === ' ') {

    if (captureActive) {

      eel.stop\_video\_feed();

      captureActive = false;

    } else {

      eel.restart\_video\_feed();

      captureActive = true;

    }

  }

});

eel.expose(get\_Option)

function get\_Option() {

  selectedOption = $('#idOption').val()

  return selectedOption;

}

eel.expose(get\_Value)

function get\_Value(*id*) {

  selectedVal= document.getElementById(*id*).innerHTML

  return selectedVal;

}

main.py:

*import* logging

*import* sys

*from* tkinter *import* Tk, messagebox

*import* eel

*import* base64

*import* time

*import* os

*import* json

*import* cv2

*import* numpy *as* np

*from* camera *import* VideoCamera

x = "null"

stop\_feed = True

detect = False

hide\_background = False

*# Set name of Video file to open. Leave name "" to open camera*

video\_name = "./web/image/car.mp4"

video\_1 = "./web/image/brick1\_05.mp4"

video\_2 = "./web/image/brick3\_11.mp4"

video\_3 = "./web/image/brick6\_18.mp4"

*# video\_name = ""*

*# Read Images*

img = cv2.imread("./web/image/empty.png",cv2.IMREAD\_GRAYSCALE)

img = cv2.resize(img,(640, 480), *interpolation* = cv2.INTER\_CUBIC)

*# Setup the images to display in html file*

@eel.expose

def setup():

   img\_send\_to\_js(img, "output")

   imgblock = cv2.imread("./web/image/green2x8.png",cv2.IMREAD\_ANYCOLOR)

   imgblock = cv2.resize(imgblock, (0, 0), *fx*=0.35, *fy*=0.35)

   img\_send\_to\_js(imgblock, "green2x8img")

   imgblock = cv2.imread("./web/image/Grey16.png",cv2.IMREAD\_ANYCOLOR)

   imgblock = cv2.resize(imgblock, (0, 0), *fx*=0.35, *fy*=0.35)

   img\_send\_to\_js(imgblock, "darkgray2x8img")

   imgblock = cv2.imread("./web/image/lightgray2x4.png",cv2.IMREAD\_ANYCOLOR)

   imgblock = cv2.resize(imgblock, (0, 0), *fx*=0.35, *fy*=0.35)

   img\_send\_to\_js(imgblock, "lightgray2x4img")

   imgblock = cv2.imread("./web/image/lime2x2.png",cv2.IMREAD\_ANYCOLOR)

   imgblock = cv2.resize(imgblock, (0, 0), *fx*=0.35, *fy*=0.35)

   img\_send\_to\_js(imgblock, "lime2x2img")

   imgblock = cv2.imread("./web/image/lime2x4.png",cv2.IMREAD\_ANYCOLOR)

   imgblock = cv2.resize(imgblock, (0, 0), *fx*=0.35, *fy*=0.35)

   img\_send\_to\_js(imgblock, "lime2x4img")

   imgblock = cv2.imread("./web/image/mediumazure4x6.png",cv2.IMREAD\_ANYCOLOR)

   imgblock = cv2.resize(imgblock, (0, 0), *fx*=0.35, *fy*=0.35)

   img\_send\_to\_js(imgblock, "mediumazure4x6img")

   imgblock = cv2.imread("./web/image/mediumblue2x4.png",cv2.IMREAD\_ANYCOLOR)

   imgblock = cv2.resize(imgblock, (0, 0), *fx*=0.35, *fy*=0.35)

   img\_send\_to\_js(imgblock, "mediumblue2x4img")

   imgblock = cv2.imread("./web/image/orange2x2.png",cv2.IMREAD\_ANYCOLOR)

   imgblock = cv2.resize(imgblock, (0, 0), *fx*=0.35, *fy*=0.35)

   img\_send\_to\_js(imgblock, "orange2x2img")

   imgblock = cv2.imread("./web/image/orange2x4.png",cv2.IMREAD\_ANYCOLOR)

   imgblock = cv2.resize(imgblock, (0, 0), *fx*=0.35, *fy*=0.35)

   img\_send\_to\_js(imgblock, "orange2x4img")

   imgblock = cv2.imread("./web/image/yellow2x2.png",cv2.IMREAD\_ANYCOLOR)

   imgblock = cv2.resize(imgblock, (0, 0), *fx*=0.35, *fy*=0.35)

   img\_send\_to\_js(imgblock, "yellow2x2img")

   imgblock = cv2.imread("./web/image/yellow2x3.png",cv2.IMREAD\_ANYCOLOR)

   imgblock = cv2.resize(imgblock, (0, 0), *fx*=0.35, *fy*=0.35)

   img\_send\_to\_js(imgblock, "yellow2x3img")

   imgblock = cv2.imread("./web/image/yellow2x4.png",cv2.IMREAD\_ANYCOLOR)

   imgblock = cv2.resize(imgblock, (0, 0), *fx*=0.35, *fy*=0.35)

   img\_send\_to\_js(imgblock, "yellow2x4img")

*#  Your code depend on image processing*

*# This is a sample code to change*

*# and send processed image to JavaScript*

@eel.expose

def video\_feed():

  global stop\_feed

*if* stop\_feed:

    stop\_feed = False

    img\_send\_to\_js("", "output")

    option= eel.get\_Option()()

*if* option:

      global x

      global video\_name

*if* option == '1':

        video\_name = video\_1

*elif* option == '2':

        video\_name = video\_2

*elif* option == '3':

        video\_name = video\_3

      x = VideoCamera(video\_name)

      y= process(x)

      text\_send\_to\_js("Video Started", "process")

      kernel = np.ones((7,7),np.uint8)

*for* frame *in* y:

        countgreen2x8 = 0

        countdarkgray2x8 = 0

        countlightgray2x4 = 0

        countlime2x2 = 0

        countlime2x4 = 0

        countmediumazure4x6 = 0

        countmediumblue2x4 = 0

        countorange2x2 = 0

        countorange2x4 = 0

        countyellow2x2 = 0

        countyellow2x3 = 0

        countyellow2x4 = 0

        newframe = frame.copy()

*#Color detection*

        hsv\_frame = cv2.cvtColor(newframe, cv2.COLOR\_BGR2HSV)

*# Azure color*

        low\_azure = np.array([180\*188/360, 255\*0.15, 255\*0.38])

        high\_azure = np.array([180\*212/360, 255\*0.75, 255\*0.78])

        azure\_mask = cv2.inRange(hsv\_frame, low\_azure, high\_azure)

        azure\_mask = cv2.morphologyEx(azure\_mask, cv2.MORPH\_CLOSE, kernel)

        azure = cv2.bitwise\_and(newframe, newframe, *mask*=azure\_mask)

*# Darkgray color #light gray color*

        low\_gray = np.array([180\*0/360, 255\*0.00, 255\*0.20])

        high\_gray = np.array([180\*360/360, 255\*0.15, 255\*0.58])

        gray\_mask = cv2.inRange(hsv\_frame, low\_gray, high\_gray)

        gray\_mask = cv2.morphologyEx(gray\_mask, cv2.MORPH\_CLOSE, kernel)

        gray = cv2.bitwise\_and(newframe, newframe, *mask*=gray\_mask)

*# Green color*

        low\_green = np.array([180\*82/360, 255\*0.15, 255\*0.23])

        high\_green = np.array([180\*145/360, 255\*0.60, 255\*0.55])

        green\_mask = cv2.inRange(hsv\_frame, low\_green, high\_green)

        green\_mask = cv2.morphologyEx(green\_mask, cv2.MORPH\_CLOSE, kernel)

        green = cv2.bitwise\_and(newframe, newframe, *mask*=green\_mask)

*# Blue color*

        low\_blue = np.array([180\*205/360, 255\*0.15, 255\*0.30])

        high\_blue = np.array([180\*250/360, 255\*0.60, 255\*0.70])

        blue\_mask = cv2.inRange(hsv\_frame, low\_blue, high\_blue)

        blue\_mask = cv2.morphologyEx(blue\_mask, cv2.MORPH\_CLOSE, kernel)

        blue = cv2.bitwise\_and(newframe, newframe, *mask*=blue\_mask)

*# Orange color*

        low\_orange = np.array([180\*0/360, 255\*0.55, 255\*0.57])

        high\_orange = np.array([180\*27/360, 255\*0.88, 255\*0.82])

        orange\_mask = cv2.inRange(hsv\_frame, low\_orange, high\_orange)

        orange\_mask = cv2.morphologyEx(orange\_mask, cv2.MORPH\_CLOSE, kernel)

        orange = cv2.bitwise\_and(newframe, newframe, *mask*=orange\_mask)

*# Lime color*

        low\_lime = np.array([180\*58/360, 255\*0.25, 255\*0.30])

        high\_lime = np.array([180\*82/360, 255\*0.78, 255\*0.70])

        lime\_mask = cv2.inRange(hsv\_frame, low\_lime, high\_lime)

        lime\_mask = cv2.morphologyEx(lime\_mask, cv2.MORPH\_CLOSE, kernel)

        lime = cv2.bitwise\_and(newframe, newframe, *mask*=lime\_mask)

*# Yellow color*

        low\_yellow = np.array([180\*34/360, 255\*0.40, 255\*0.55])

        high\_yellow = np.array([180\*54/360, 255\*0.90, 255\*0.82])

        yellow\_mask = cv2.inRange(hsv\_frame, low\_yellow, high\_yellow)

        yellow\_mask = cv2.morphologyEx(yellow\_mask, cv2.MORPH\_CLOSE, kernel)

        yellow = cv2.bitwise\_and(newframe, newframe, *mask*=yellow\_mask)

*# Combined colors*

*if* hide\_background:

          combined\_mask = azure\_mask + gray\_mask + green\_mask + blue\_mask + orange\_mask + lime\_mask + yellow\_mask

          newframe = cv2.bitwise\_and(newframe, newframe, *mask*=combined\_mask)

*#Detection TOGGLE*

*if* detect:

*#Green block detect*

          cframe = cv2.cvtColor(green,cv2.COLOR\_BGR2GRAY)

          cframe = cv2.GaussianBlur(cframe,(3,3),0)

          circles = cv2.HoughCircles(cframe,cv2.HOUGH\_GRADIENT,1,24,*param1*=50,*param2*=8,*minRadius*=3,*maxRadius*=8)

          greengrayed = cv2.cvtColor(green, cv2.COLOR\_BGR2GRAY)

          contours,hierarchy=cv2.findContours(greengrayed,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)

*for* pic, contour *in* enumerate(contours):

            area = cv2.contourArea(contour)

*if*(area>2500):

              countc = 0

              x,y,w,h = cv2.boundingRect(contour)

*if* circles is not None:

                circles = np.uint16(np.around(circles))

*for* i *in* circles[0,:]:

*if*(x<i[0] and i[0]<(x+w) and y<i[1] and i[1]<(y+h)):

*# draw the circle*

                      cv2.circle(newframe,(i[0],i[1]),i[2],(255,0,200),2)

                      countc=countc+1

*#newframe = cv2.rectangle(newframe,(x,y),(x+w,y+h),(0,0,255),2)*

              rect = cv2.minAreaRect(contour)

              box = cv2.boxPoints(rect)

              box = np.int0(box)

              newframe = cv2.drawContours(newframe,[box],0,(0,255,0),2)

              countgreen2x8 += 1

              cv2.putText(newframe,"green2x8",(x,y-8),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))

*#cv2.putText(newframe,str(countc) + " " + str(area),(x,y-18),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))*

*#Light and Dark gray block detect*

          cframe = cv2.cvtColor(gray,cv2.COLOR\_BGR2GRAY)

          cframe = cv2.GaussianBlur(cframe,(3,3),0)

          circles = cv2.HoughCircles(cframe,cv2.HOUGH\_GRADIENT,1,24,*param1*=80,*param2*=8,*minRadius*=3,*maxRadius*=8)

          graygrayed = cv2.cvtColor(gray, cv2.COLOR\_BGR2GRAY)

          contours,hierarchy=cv2.findContours(graygrayed,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)

*for* pic, contour *in* enumerate(contours):

            area = cv2.contourArea(contour)

*if*(area>5000):

              countc = 0

              x,y,w,h = cv2.boundingRect(contour)

*if* circles is not None:

                circles = np.uint16(np.around(circles))

*for* i *in* circles[0,:]:

*if*(x<i[0] and i[0]<(x+w) and y<i[1] and i[1]<(y+h)):

*# draw the circle*

                      cv2.circle(newframe,(i[0],i[1]),i[2],(255,0,200),2)

                      countc=countc+1

*#newframe = cv2.rectangle(newframe,(x,y),(x+w,y+h),(0,0,255),2)*

              rect = cv2.minAreaRect(contour)

              box = cv2.boxPoints(rect)

              box = np.int0(box)

              newframe = cv2.drawContours(newframe,[box],0,(0,255,0),2)

*if* countc > 10:

                countdarkgray2x8 += 1

                cv2.putText(newframe,"Gray16",(x,y-8),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))

*else*:

                countlightgray2x4 += 1

                cv2.putText(newframe,"lightgray2x4",(x,y-8),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))

*#cv2.putText(newframe,str(countc) + " " + str(area),(x,y-18),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))*

*#Lime block detect*

          cframe = cv2.cvtColor(lime,cv2.COLOR\_BGR2GRAY)

          cframe = cv2.GaussianBlur(cframe,(3,3),0)

          circles = cv2.HoughCircles(cframe,cv2.HOUGH\_GRADIENT,1,24,*param1*=70,*param2*=9,*minRadius*=3,*maxRadius*=8)

          limegrayed = cv2.cvtColor(lime, cv2.COLOR\_BGR2GRAY)

          contours,hierarchy=cv2.findContours(limegrayed,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)

*for* pic, contour *in* enumerate(contours):

            area = cv2.contourArea(contour)

*if*(area>2500):

              countc = 0

              x,y,w,h = cv2.boundingRect(contour)

*if* circles is not None:

                circles = np.uint16(np.around(circles))

*for* i *in* circles[0,:]:

*if*(x<i[0] and i[0]<(x+w) and y<i[1] and i[1]<(y+h)):

*# draw the circle*

                      cv2.circle(newframe,(i[0],i[1]),i[2],(255,0,200),2)

                      countc=countc+1

*#newframe = cv2.rectangle(newframe,(x,y),(x+w,y+h),(0,0,255),2)*

              rect = cv2.minAreaRect(contour)

              box = cv2.boxPoints(rect)

              box = np.int0(box)

              newframe = cv2.drawContours(newframe,[box],0,(0,255,0),2)

*if* countc > 4:

                countlime2x4 += 1

                cv2.putText(newframe,"lime2x4",(x,y-8),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))

*else*:

                countlime2x2 += 1

                cv2.putText(newframe,"lime2x2",(x,y-8),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))

*#cv2.putText(newframe,str(countc) + " " + str(area),(x,y-18),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))*

*#Azure block detect*

          cframe = cv2.cvtColor(azure,cv2.COLOR\_BGR2GRAY)

          cframe = cv2.GaussianBlur(cframe,(3,3),0)

          circles = cv2.HoughCircles(cframe,cv2.HOUGH\_GRADIENT,1,24,*param1*=60,*param2*=7,*minRadius*=3,*maxRadius*=8)

          azuregrayed = cv2.cvtColor(azure, cv2.COLOR\_BGR2GRAY)

          contours,hierarchy=cv2.findContours(azuregrayed,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)

*for* pic, contour *in* enumerate(contours):

            area = cv2.contourArea(contour)

*if*(area>2500):

              countc = 0

              x,y,w,h = cv2.boundingRect(contour)

*if* circles is not None:

                circles = np.uint16(np.around(circles))

*for* i *in* circles[0,:]:

*if*(x<i[0] and i[0]<(x+w) and y<i[1] and i[1]<(y+h)):

*# draw the circle*

                      cv2.circle(newframe,(i[0],i[1]),i[2],(255,0,200),2)

                      countc=countc+1

*#newframe = cv2.rectangle(newframe,(x,y),(x+w,y+h),(0,0,255),2)*

              rect = cv2.minAreaRect(contour)

              box = cv2.boxPoints(rect)

              box = np.int0(box)

              newframe = cv2.drawContours(newframe,[box],0,(0,255,0),2)

              countmediumazure4x6 += 1

              cv2.putText(newframe,"mediumazure4x6",(x,y-8),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))

*#cv2.putText(newframe,str(countc) + " " + str(area),(x,y-18),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))*

*#Blue block detect*

          cframe = cv2.cvtColor(blue,cv2.COLOR\_BGR2GRAY)

          cframe = cv2.GaussianBlur(cframe,(3,3),0)

          circles = cv2.HoughCircles(cframe,cv2.HOUGH\_GRADIENT,1,24,*param1*=70,*param2*=9,*minRadius*=3,*maxRadius*=8)

          bluegrayed = cv2.cvtColor(blue, cv2.COLOR\_BGR2GRAY)

          contours,hierarchy=cv2.findContours(bluegrayed,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)

*for* pic, contour *in* enumerate(contours):

            area = cv2.contourArea(contour)

*if*(area>2500):

              countc = 0

              x,y,w,h = cv2.boundingRect(contour)

*if* circles is not None:

                circles = np.uint16(np.around(circles))

*for* i *in* circles[0,:]:

*if*(x<i[0] and i[0]<(x+w) and y<i[1] and i[1]<(y+h)):

*# draw the circle*

                      cv2.circle(newframe,(i[0],i[1]),i[2],(255,0,200),2)

                      countc=countc+1

*#newframe = cv2.rectangle(newframe,(x,y),(x+w,y+h),(0,0,255),2)*

              rect = cv2.minAreaRect(contour)

              box = cv2.boxPoints(rect)

              box = np.int0(box)

              newframe = cv2.drawContours(newframe,[box],0,(0,255,0),2)

              countmediumblue2x4 += 1

              cv2.putText(newframe,"mediumblue2x4",(x,y-8),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))

*#cv2.putText(newframe,str(countc) + " " + str(area),(x,y-18),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))*

*#Orange block detect*

          cframe = cv2.cvtColor(orange,cv2.COLOR\_BGR2GRAY)

          cframe = cv2.GaussianBlur(cframe,(3,3),0)

          circles = cv2.HoughCircles(cframe,cv2.HOUGH\_GRADIENT,1,24,*param1*=50,*param2*=8,*minRadius*=3,*maxRadius*=8)

          orangegrayed = cv2.cvtColor(orange, cv2.COLOR\_BGR2GRAY)

          contours,hierarchy=cv2.findContours(orangegrayed,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)

*for* pic, contour *in* enumerate(contours):

            area = cv2.contourArea(contour)

*if*(area>2500):

              countc = 0

              x,y,w,h = cv2.boundingRect(contour)

*if* circles is not None:

                circles = np.uint16(np.around(circles))

*for* i *in* circles[0,:]:

*if*(x<i[0] and i[0]<(x+w) and y<i[1] and i[1]<(y+h)):

*# draw the circle*

                      cv2.circle(newframe,(i[0],i[1]),i[2],(255,0,200),2)

                      countc=countc+1

*#newframe = cv2.rectangle(newframe,(x,y),(x+w,y+h),(0,0,255),2)*

              rect = cv2.minAreaRect(contour)

              box = cv2.boxPoints(rect)

              box = np.int0(box)

              newframe = cv2.drawContours(newframe,[box],0,(0,255,0),2)

*if* countc > 4:

                countorange2x4 += 1

                cv2.putText(newframe,"orange2x4",(x,y-8),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))

*else*:

                countorange2x2 += 1

                cv2.putText(newframe,"orange2x2",(x,y-8),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))

*#cv2.putText(newframe,str(countc) + " " + str(area),(x,y-18),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))*

*#Yellow block detect*

          cframe = cv2.cvtColor(yellow,cv2.COLOR\_BGR2GRAY)

*#cframe = cv2.GaussianBlur(cframe,(3,3),0)*

          circles = cv2.HoughCircles(cframe,cv2.HOUGH\_GRADIENT,1,24,*param1*=40,*param2*=9,*minRadius*=3,*maxRadius*=8)

          yellowgrayed = cv2.cvtColor(yellow, cv2.COLOR\_BGR2GRAY)

          contours,hierarchy=cv2.findContours(yellowgrayed,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)

*for* pic, contour *in* enumerate(contours):

            area = cv2.contourArea(contour)

*if*(area>2500):

              countc = 0

              x,y,w,h = cv2.boundingRect(contour)

*if* circles is not None:

                circles = np.uint16(np.around(circles))

*for* i *in* circles[0,:]:

*if*(x<i[0] and i[0]<(x+w) and y<i[1] and i[1]<(y+h)):

*# draw the circle*

                      cv2.circle(newframe,(i[0],i[1]),i[2],(255,0,200),2)

                      countc=countc+1

*#newframe = cv2.rectangle(newframe,(x,y),(x+w,y+h),(0,0,255),2)*

              rect = cv2.minAreaRect(contour)

              box = cv2.boxPoints(rect)

              box = np.int0(box)

              newframe = cv2.drawContours(newframe,[box],0,(0,255,0),2)

*if* countc > 6:

                countyellow2x4 += 1

                cv2.putText(newframe,"yellow2x4",(x,y-8),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))

*elif* countc > 4:

                countyellow2x3 += 1

                cv2.putText(newframe,"yellow2x3",(x,y-8),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))

*else*:

                countyellow2x2 += 1

                cv2.putText(newframe,"yellow2x2",(x,y-8),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))

*#cv2.putText(newframe,str(countc) + " " + str(area),(x,y-18),cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,0,255))*

        img\_send\_to\_js(newframe, "output")

        text\_send\_to\_js(countgreen2x8, "green2x8count")

        text\_send\_to\_js(countdarkgray2x8, "darkgray2x8count")

        text\_send\_to\_js(countlightgray2x4, "lightgray2x4count")

        text\_send\_to\_js(countlime2x2, "lime2x2count")

        text\_send\_to\_js(countlime2x4, "lime2x4count")

        text\_send\_to\_js(countmediumazure4x6, "mediumazure4x6count")

        text\_send\_to\_js(countmediumblue2x4, "mediumblue2x4count")

        text\_send\_to\_js(countorange2x2, "orange2x2count")

        text\_send\_to\_js(countorange2x4, "orange2x4count")

        text\_send\_to\_js(countyellow2x2, "yellow2x2count")

        text\_send\_to\_js(countyellow2x3, "yellow2x3count")

        text\_send\_to\_js(countyellow2x4, "yellow2x4count")

        total\_blocks = countgreen2x8 + countdarkgray2x8 + countlightgray2x4 + countlime2x2 + countlime2x4 + countmediumazure4x6 + countmediumblue2x4 + countorange2x2 + countorange2x4 + countyellow2x2 + countyellow2x3 + countyellow2x4

        text\_send\_to\_js(total\_blocks, "totalcount")

*#time.sleep(0.01)*

      text\_send\_to\_js("Video Ended", "process")

*else*:

      text\_send\_to\_js("Select Option before pressing Start Video", "process")

      img\_send\_to\_js(img, "output")

      stop\_feed = True

*else*:

    text\_send\_to\_js("Please stop the current video first before starting another", "process")

*#  Your code depend on image processing*

*# This is a sample code to change to change*

*# the text and send the text to JavaScript*

@eel.expose

def toggle\_detect():

  global detect

  detect = not detect

@eel.expose

def toggle\_background():

  global hide\_background

  hide\_background = not hide\_background

*# Get Camera from video feed*

*# Add ur codes to process here*

def process(*camera*):

  option= eel.get\_Option()()

  text\_send\_to\_js("Option Selected: Level " + str(option), "p1")

  global stop\_feed

*while* (True and not stop\_feed):

    success, frame = camera.get\_frame()

    grab, lastframe =camera.get\_frame()

*if* not grab:

      stop\_feed = True

*break*

*yield* frame

*# Stop Video Caturing*

*# Do not touch*

@eel.expose

def stop\_video\_feed():

  global stop\_feed

  stop\_feed = True

*#x.stop\_capturing()*

  text\_send\_to\_js("Video Stopped", "p2")

*# Restart Video Caturing*

*# Do not touch*

@eel.expose

def restart\_video\_feed():

  stop\_video\_feed()

  video\_feed()

*#x.restart\_capturing()*

  text\_send\_to\_js("Video restarted", "p2")

*# Send text from python to Javascript*

*# Do not touch*

def text\_send\_to\_js(*val*,*id*):

  eel.updateTextSrc(val,id)()

*# Send image from python to Javascript*

*# Do not touch*

def img\_send\_to\_js(*img*, *id*):

*if* np.shape(img) == () :

    eel.updateImageSrc("", id)()

*else*:

    ret, jpeg = cv2.imencode(".jpg",img)

    jpeg.tobytes()

    blob = base64.b64encode(jpeg)

    blob = blob.decode("utf-8")

    eel.updateImageSrc(blob, id)()

*# Start function for app*

*# Do not touch*

def start\_app():

*try*:

    start\_html\_page = 'index.html'

    eel.init('web')

    logging.info("App Started")

    eel.start('index.html', *size*=(1000, 800))

*except* Exception *as* e:

    err\_msg = 'Could not launch a local server'

    logging.error('{}\n{}'.format(err\_msg, e.args))

    show\_error(*title*='Failed to initialise server', *msg*=err\_msg)

    logging.info('Closing App')

    sys.exit()

*if* \_\_name\_\_ == "\_\_main\_\_":

  x = VideoCamera(video\_name)

  start\_app()