ESDII Lab 1- Tennis Ball Detection: RGB Filter With Green & Blue Tennis Ball Detection

Aliana Tejeda *CPET*

I. Introduction

THE purpose of this lab was to create an embedded tennis ball tracker that has the ability to load an image and filter the image for different tennis ball colors. In addition to filtering for various other colors with using the upper and lower bounds of RBG, there is also the capability of filtering with brightness and contrast. The interface is built with using PyQt4 and the filtering of the images is done with openCv2's inRange, erode and dilate methods.

II. Filter

A. Getting the Values to Filter

The filtering of the image is done by the QtGui sliders instantiated in **calibration.py** and the filter method in **faceTracker.py**. The file **globals.py** has a class that holds the variables which are being used to filter the image that will be loaded by the file menu. The file **calibration.py** instantiates sliders for each and every filtering variable, sets a default value to the sliders, and a method for each slider, so for every event of the slider, it is handled and evaluated as such. **calibration.py** updates the values of the global variables of **globals.py** by the corresponding update methods for each slider. **calibration.py** and **globals.py** do a lot of handshaking in terms of passing variable values and updating variables.

B. Loading and Filtering

The interface for the program allows for the user to upload any image that has the extension of ".jpg". After uploading the image. The program waits for the user to enable the "Filter Activation" button to start filtering the image. The program does not automatically filter images as soon as the program is started. The user has to click the "Filter Activation" button in order to enable to the filtering to take place. There are also pre-configured parameters files that will filter to detect a blue or green tennis ball. These two parameters are loaded and enabled based on if the radio button corresponding to the desired filter is checked.

C. Passing Image File Path

The program is equipped with a file menu that allows the user to upload files. This feature is located in **top.py**. This feature is a simple drop-down menu that only has the action of "upload image". The file menu feature itself has a method that handles when the user clicks the "upload image" action. The method is invoked after the user clicks to upload an image, and the method will open the file explorer for the user and depending where the images are stored, the user will navigate to the image and open it. The method then takes the path of the image the user selected and passes it to "image_filepath" which is part of **globals.py** class.

D. Filtering Loaded Image

The crux of the filtering process takes place in the **faceTracker.py** file. **faceTracker.py** has a timer event, and for each time the timer reaches its count, it calls the *timerEvent* method which reads and parses the data read from the "image_filepath". After it reads the file path, it calls another method named "image_data_slot" which dependent on the "Filter Activation" state, will either filter the image and then parse it to the widget window, or not filter the image and just parse it to the window. While filtering, the program will draw a circle around the object it was able to find with a centroid to rest exactly where and what it found with the applied filters. The program will display the coordinates of the centroid at the top left-hand corner of the image so that the user has a better sense as to where the centroid really is in on the image.

E. Conclusion

The program is able to effectively filter for different colored tennis balls, load multiple different images, filter for specifically green or blue tennis balls by a radio option, and display the result of a filter on the image by placing a centroid on the object is found and circle the image it found. However, when it comes to filtering these different colored tennis balls. Brightness and contrast have little to no effect in the process of filtering the image. The upper and lower bounds of the RGB sections are more effective in the filtering process.

III. Code Discussion

Invokes the layouts of the two files to be compiled

```
        # instantiate the dock widgets
    self.faceTracker = FaceTrackerHolder(self)
    self.calibration = Calibration(self)
```

Sets the default values of the filter variables to a number so that the lower bounds and upper bounds are not equal and throw an assertion error

```
self.brightness.setValue(1)
self.rMin.setValue(1)
self.rMax.setValue(100)
self.bMin.setValue(1)
self.bMax.setValue(100)
self.gMin.setValue(1)
self.gMax.setValue(100)
self.Contrast.setValue(1)
```

Method for "Filter Activation", that when invoked will update the global filter variables and set the filter condition to 1 to allow the method in *facaeTracker.py imabe_data_slot to filter the image*

```
def filters(self, pressed):
    if pressed:
        self.brightness.setValue(globals.Brightness)
        self.Contrast.setValue(globals.contrast)
        self.rMin.setValue(globals.rMin)
        self.rMax.setValue(globals.rMax)
        self.bMin.setValue(globals.bMin)
        self.bMax.setValue(globals.bMax)
        self.gMin.setValue(globals.gMin)
        self.gMax.setValue(globals.gMax)
        globals.filterOption = 1
    pass
```

The method that handles the event when the user clicks the radio button for blue detection or green detection of a tennis ball

```
def btnstate(self, b):
    if b.text() == "Blue Detection":
        if b.isChecked():
            with open("bluedetection.txt") as f:
                globals.Brightness = int(f.readline().split("= ")[1])
                globals.contrast = float(f.readline().split("= ")[1])
                globals.rMin = float(f.readline().split("= ")[1])
                globals.rMax = float(f.readline().split("= ")[1])
                globals.bMin = float(f.readline().split("= ")[1])
                globals.bMax = float(f.readline().split("= ")[1])
                globals.gMin = float(f.readline().split("= ")[1])
                globals.gMax = float(f.readline().split("= ")[1])
            self.brightness.setValue(globals.Brightness)
            self.Contrast.setValue(globals.contrast)
            self.rMin.setValue(globals.rMin)
            self.rMax.setValue(globals.rMax)
            self.bMin.setValue(globals.bMin)
            self.bMax.setValue(globals.bMax)
            self.gMin.setValue(globals.gMin)
            self.gMax.setValue(globals.gMax)
        else:
            pass

    if b.text() == "Green Detection":
        if b.isChecked():
            with open("greendetection.txt") as f:
                globals.Brightness = int(f.readline().split("= ")[1])
                globals.contrast = float(f.readline().split("= ")[1])
                globals.rMin = float(f.readline().split("= ")[1])
                globals.rMax = float(f.readline().split("= ")[1])
                globals.bMin = float(f.readline().split("= ")[1])
                globals.bMax = float(f.readline().split("= ")[1])
                globals.gMin = float(f.readline().split("= ")[1])
                globals.gMax = float(f.readline().split("= ")[1])
            self.brightness.setValue(globals.Brightness)
            self.Contrast.setValue(globals.contrast)
            self.rMin.setValue(globals.rMin)
            self.rMax.setValue(globals.rMax)
            self.bMin.setValue(globals.bMin)
            self.bMax.setValue(globals.bMax)
            self.gMin.setValue(globals.gMin)
            self.gMax.setValue(globals.gMax)
        else:
            pass
```

Lines of code that create the file menu which are located in *top.py* when there is an event, sends the event to the method *processtrigger*

```
bar = self.menuBar()
file = bar.addMenu("File")
file.addAction("Upload")
file.triggered[QAction].connect(self.processtrigger)
```

This is the method called by the file menu to open the file explorer so that the user can upload different images to be filtered

```
def processtrigger(self):
    fileName = QtGui.QFileDialog.getOpenFileName(None, "Enter Filename.", ".jpg", "(*.jpg)")
    if not fileName:
        print('Could not open or find the image: ', str(fileName))
        pass
    else:
        globals.image_filepath = fileName
```

The class that resides in *globals.py* which holds the global variables that are used in *calaibration.py, faceTracker.py,* and *top.py*

```
class globals():
    rMin = 100
    rMax = 100
    gMin = 100
    gMax = 100
    bMin = 100
    bMax = 100
    Brightness = 2
    contrast = 25
    image_filepath = "../images/loadfile.jpg"
    filterOption = 0
```

This is the timerEvent method that is called repeatedly when the timer reaches its count. "image_filepath" is constantly checked for updates and updates the image being displayed. There is a conditional to make sure that in case there is nothing at the file path, it does not try to read None type

```
def timerEvent(self, event):
    if (event.timerId() != self.timer.timerId()):
        return
    data = cv2.imread(str(globals.image_filepath))
    if data is None:
        pass
    else:
        self.image_data.emit(data)
```

This is the method *image_data_slot* which based on the "Filter Activation" condition, will either not filter the image, or filter it.

```
# this function is called from self.image_data.emit(data) in line 30 when the timer ticks
def image_data_slot(self, image_data):
    if globals.filterOption == 0:
        self.image = self.get_qimage(image_data)
    else:
        img = cv2.imread(str(globals.image_filepath))
        data = cv2.resize(img,(640,480))

        # get info from track bar and apply to result
        rMin = globals.rMin
        rMax = globals.rMax
        gMin = globals.gMin
        gMax = globals.gMax
        bMin = globals.bMin
        bMax = globals.bMax
        alpha = globals.Brightness
        beta = globals.contrast
        # generate threshold array
        lower = np.array([bMin, gMin, rMin])
        upper = np.array([bMax, gMax, rMax])

        mask = cv2.inRange(data, lower, upper)
        mask = cv2.erode(mask, None, iterations=2)
        mask = cv2.dilate(mask, None, iterations=2)

        new_image = cv2.convertScaleAbs(data, alpha=alpha, beta=beta)
        maskedImage = cv2.bitwise_and(data, new_image, mask=mask)

        clone_img = copy.copy(data)

        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]
        center = None
        # only proceed if at least one contour was found
        if len(cnts) > 0:
            # find the largest contour in the mask, then use
            # it to compute the minimum enclosing circle and
            # centroid
            c = max(cnts, key=cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(c)
            M = cv2.moments(c)
            center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
```

```
    cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]
    center = None
    # only proceed if at least one contour was found
    if len(cnts) > 0:
        # find the largest contour in the mask, then use
        # it to compute the minimum enclosing circle and
        # centroid
        c = max(cnts, key=cv2.contourArea)
        ((x, y), radius) = cv2.minEnclosingCircle(c)
        M = cv2.moments(c)
        center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

        # only proceed if the radius meets a minimum size
        if radius > 10:
            # draw the circle and centroid on the frame,
            # then update the list of tracked points
            cv2.circle(clone_img, (int(x), int(y)), int(radius), (0, 255, 255), 2)
            cv2.circle(clone_img, center, 5, (0, 0, 255), -1)

             # font
            font = cv2.FONT_HERSHEY_SIMPLEX

            # org
            org = (50, 50)

            # fontScale
            fontScale = 1

            # Blue color in BGR
            color = (255, 0, 0)

            # Line thickness of 2 px
            thickness = 2

            # Using cv2.putText() method
            image = cv2.putText(clone_img,'X: ' + str(round(x,2)) + " Y: " + str(round(y,2)), org, font,
                                fontScale, color, thickness, cv2.LINE_AA)
    self.image = self.get_qimage(clone_img)
    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        pass

# update() calls the paintEvent() function below
self.update()
```

The lines of code that add the functionality of the "Filter Activation", and blue green tennis ball detection.

```
self.filterOP = QtGui.QPushButton("Filter Activation", self)
self.filterOP.setCheckable(True)
self.filterOP.clicked[bool].connect(self.filters)

self.radioB = QtGui.QRadioButton("Blue Detection")
self.radioB.toggled.connect(lambda: self.btnstate(self.radioB))

self.radioG = QtGui.QRadioButton("Green Detection")
self.radioG.toggled.connect(lambda: self.btnstate(self.radioG))
```

IV. Code Flow