Object Detection

This project was made in Python 2.7 and OpenCV 3 by Christian Melendez.

Ball tracking

There are two different methods to detects the ball. One only uses the colour of the ball (*see ball_traker()*) and the other is comparing a trained ball with the input (see *ball_tracker_cascade()*).

Ball_tracker():

First, the user has different options to choose the desired colour to be detected. If you left-click, drag and release in an area, an average of the ***HSV*** values will be calculated and the trackbars will be moved to the correct position. There are two trackbars for each h, s and v, which are for the lower and upper bounds which will be used for getting the threshold image. After the drag and click, the lower and upper trackbars will be separated by 20, it is recommended to adjust the values to get a better result.

When the threshold starts to look like a ball, a circle will be detected with the function *minEnclosingCircle* from OpenCV. If the radius is big enough, the function will return the x, y and radius of the ball.

The problem with this method is that it may see other same colour objects as balls, which we don't want that. One solution is to adjust more the HSV values, but it seems to not help with the edge cases

Ball_tracker_cascade():

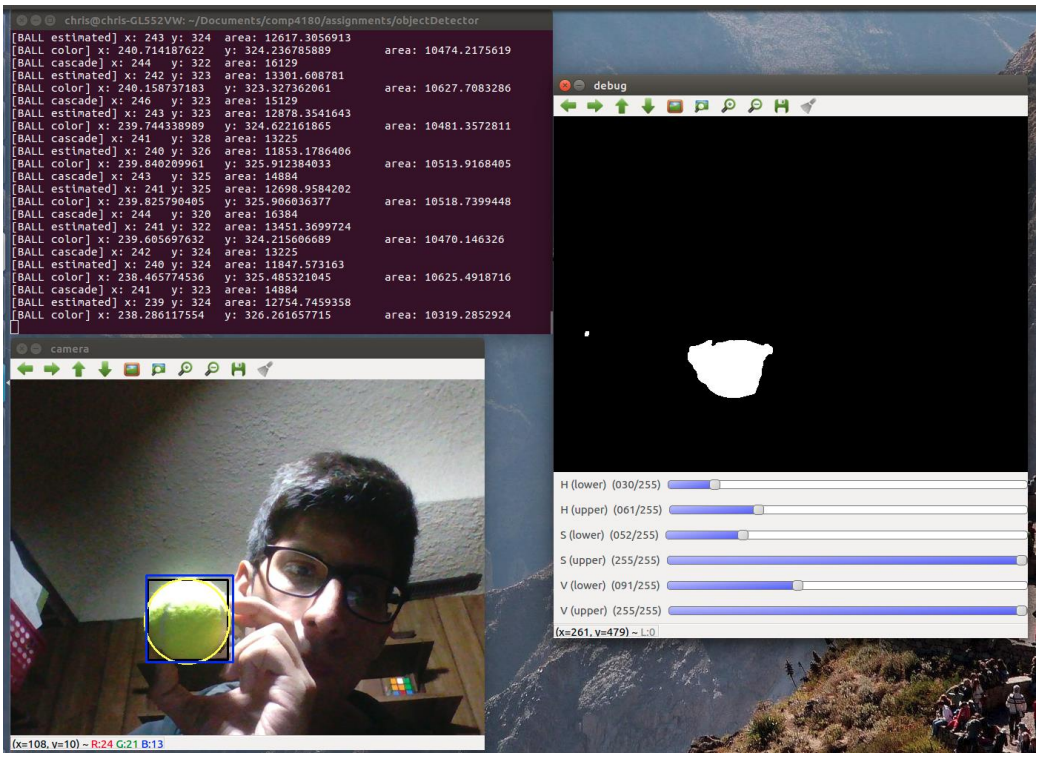
Haar cascade from OpenCV is used as an alternative to detect the ball. Pictures that show a clear tennis ball (either yellow and orange) were collected for the positives and pictures of people, false balls and same colour object were collected for the negatives. All the pictures were resized (640x480) and converted to grayscale. For the positives, a *lst* file needed to be created, this had to have the name of each picture, the number of positive objects present in the picture and the coordinates of each one. Using the ball_tracker() function, each picture was carefully double checked manually for the correct coordinates, which took some time, but the colour detection helped a lot.

After having everything ready, a vector file had to be created, also with an OpenCV function and then train it. Initially, the model was trained for 15 iterations with a hit point of 0.99, the results were not good because there was a lot of false positives. Then, the model had 30 iterations with hit point of 0.7, but the model was so accurate that couldn't detect the ball in different lighting. Finally, 20 iterations with a hit point of 0.8 was better. It still gets confused with faces and ball-like objects, but when there is a tennis ball present, it detects it well.

This function returns the x, y and area of a bounding square

In the main script, both functions are called, and their return values are compared. The orange circle is from the colour detection, the blue square from haar cascade and the black square is the average of those two. The values are also printed in console.

Also, all the values changed for the colour are saved in a YAML file, so there is no need to re-calibrate the colour from scratch



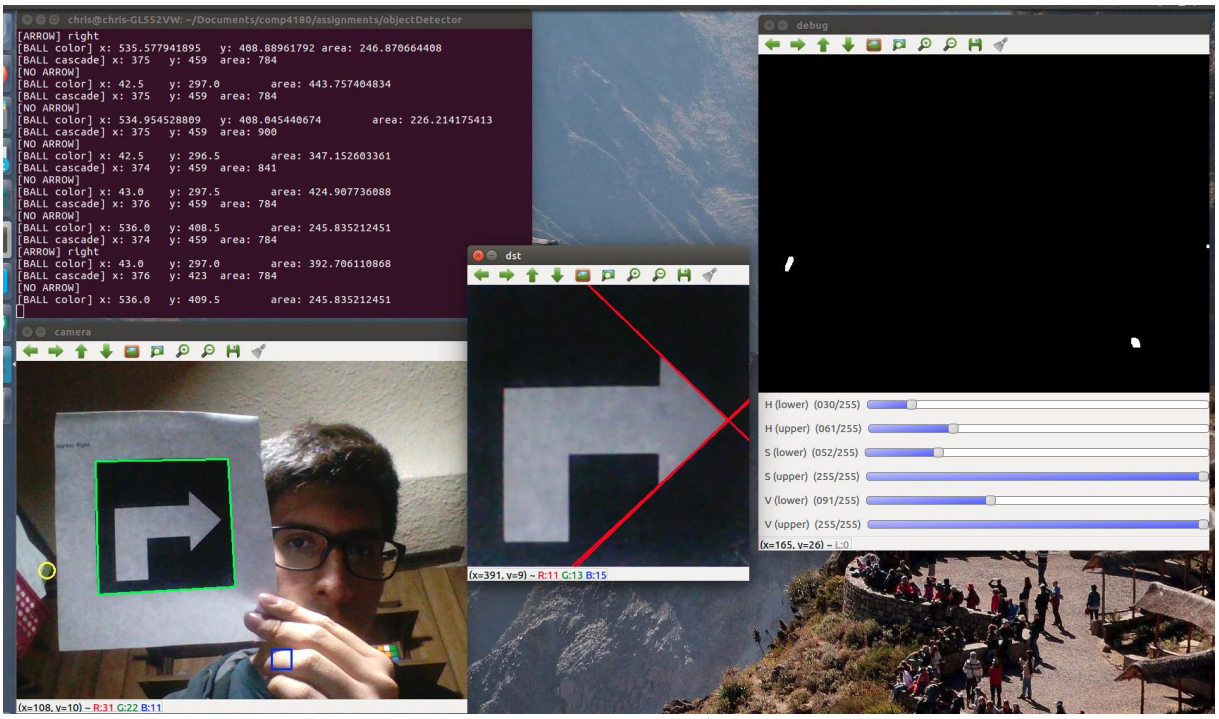
Marathon marker detector

The arrow detector has a different approach. First, a canny output is extracted from the grayscale video feedback and all the contours are found from it. In the list of all the contours, the priority is to find a 4-edge contour, which will be our marker. This may have some false positives, but this is filtered by a minimum size. After having the correct number of edges and size, the array given is sorted by the y-value. Now, because of the field of view, the arrow may be seeing from an angle, which may be difficult to detect it correctly. That is why that area is converted to a flat perspective; some of the values for conversion need to be sorted again given that the contours are not always in order.

After having the rectangular area where our candidate arrow is, canny is applied again with different values and then use houghLines from OpenCV. This will draw all the lines detected but be mostly care about the one that are diagonal. Therefore, is two diagonal lines are found, the area has a candidate arrow.

With those two lines, let's use the line-line intersection equation, which will return the parametric value of in which part these two lines collide. If they collide, then we apply parametric equation of the line to get the coordinate, compare the point with the expected direction of an arrow (forward arrow will have the point above that the center, left will be to the left side of the screen, etc.) and then the arrow type is decided.

This method works most of the time, a count of "not found" was added so the detection becomes more consecutive.


chris@chris-GL552VW: ~/Documents/comp4180/assignments/objectDetector
[ARROW] right
[BALL color] x: 535.577941895 y: 408.88961792 area: 246.870664408
[BALL cascade] x: 375 y: 459 area: 784
[NO ARROW]
[BALL color] x: 42.5 y: 297.0 area: 443.757404834
[BALL cascade] x: 375 y: 459 area: 784
[NO ARROW]
[BALL color] x: 534.954528809 y: 408.045440674 area: 226.214175413
[BALL cascade] x: 375 y: 459 area: 900
[NO ARROW]
[BALL color] x: 42.5 y: 296.5 area: 347.152603361
[BALL cascade] x: 374 y: 459 area: 841
[NO ARROW]
[BALL color] x: 43.0 y: 297.5 area: 424.907736088
[BALL cascade] x: 376 y: 459 area: 784
[NO ARROW]
[BALL color] x: 536.0 y: 408.5 area: 245.835212451
[BALL cascade] x: 374 y: 459 area: 784
[ARROW] right
[BALL color] x: 43.0 y: 297.0 area: 392.706110868
[BALL cascade] x: 376 y: 423 area: 784
[NO ARROW]
[BALL color] x: 536.0 y: 409.5 area: 245.835212451
camera
(x=108, y=10) ~ R:31 G:22 B:11
dst
(x=391, y=9) ~ R:11 G:13 B:15
debug
H (lower) (030/255)
H (upper) (061/255)
S (lower) (052/255)
S (upper) (255/255)
V (lower) (091/255)
V (upper) (255/255)
(x=165, y=26) ~ L:0