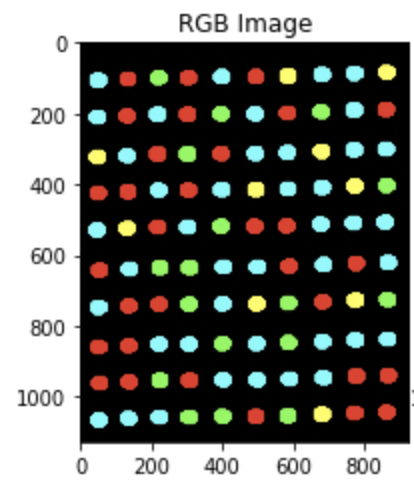
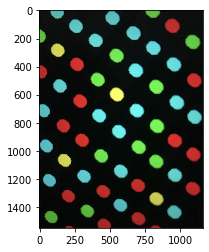
**GBMC-Part3**

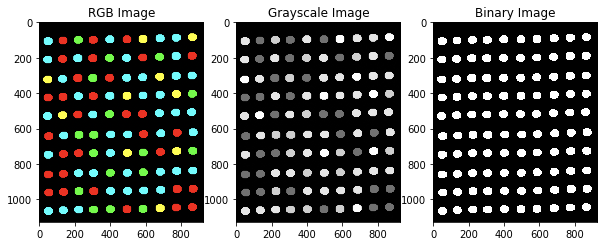
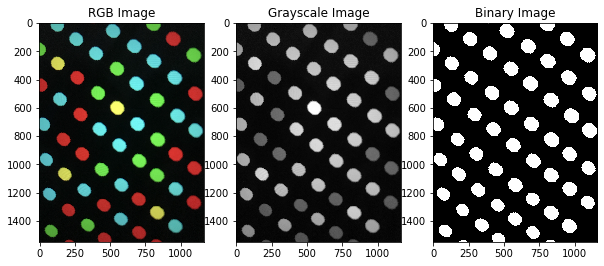
**Objectives:**

Match 3x3 color-coded patterns across two RGB images.



**Approach:**

* Convert the image to binary using a suitable threshold. Segment all circular/irregular dots in both images using 8-connectivity. (cv2.connectedComponents)



* The centroids are stored calculating the x & y mean of the labels. The colors at the centroids are predicted using mean-squared difference. The output is stored in the form of a dictionary {**centroid: color}**.

Image 1: Centroids and colors

{(26, 129): 'B', (0, 383): 'R', (24, 806): 'G', (55, 520): 'B', (112, 944): 'R'....}

* The 8 nearest neighbors of every centroid are computed using the Euclidean distance between centroids. The colors of those neighbors are stored for debugging purposes.

Image 1: Centroids and their nearest 8 centroids (Nearest Neighbors of circle at (215,401))

{… (215, 401): {(0, 383): 'R', (55, 520): 'B', (113, 261): 'B', (163, 670): 'B', (284, 131): 'Y', (322, 549): 'G', (385, 269): 'R', (493, 415): 'G'}

* These neighbors are aligned in a clockwise orientation. In this way, we can extract the 3x3 pattern for all the components. (The centroids at the corners are ignored as their 8 nearest neighbors do not classify in the customized pattern category)

Image 1: Centroids and 3x3 oriented pattern

{(26, 129): 'BGRYBRBBR', (0, 383): 'RBBYRBBGB', (24, 806): 'GBBGBBRBG', (55, 520): 'BRBBBGBBG', (112, 944): 'RGBBBBBGR', (113, 261): 'BBGYRBGBR', (163, 670): 'BRBBGBBRG', (186, 20): 'GRYBRBBBR', ….}

* With every corresponding image, we have a dictionary which has the center of all the circles and the corresponding 3x3 pattern formed around that circle.

Image 1: Centroids and 3x3 coded-pattern

{(26, 129): '320130330', (0, 383): '033103323', (24, 806): '233233032', (55, 520): '303332332', (112, 944): '023333320', (113, 261): '332103230', (163, 670): '303323302', (186, 20): '201303330', ….}

* The single pattern can be circular shifted to form 8 variants. In this way, with every corresponding circle, we get the 8 possible pattern codes associated with that circle and its neighbors.

Image 1: Centroids and all possible variants of 3x3 coded pattern

{(26, 129): ['320130330', '302013033', '330201303', '333020130', '303302013', '330330201', '313033020', '301303302'], (0, 383): ['033103323', '033310332', '023331033', '032333103', '033233310', '003323331', '010332333', '031033233'], ….}

* These codes are matched across two images. We can easily map the central point of the 3x3 pattern code from one image to another.

No. of same 3x3 patterns detected: 22

Pattern: BRGGBBRBYFound at (215, 401) in image 1Found at (843, 682) in image 2

Pattern: BRGBGBGBGFound at (268, 809) in image 1Found at (951, 498) in image 2

Pattern: GRGYBBBBBFound at (322, 549) in image 1Found at (847, 586) in image 2

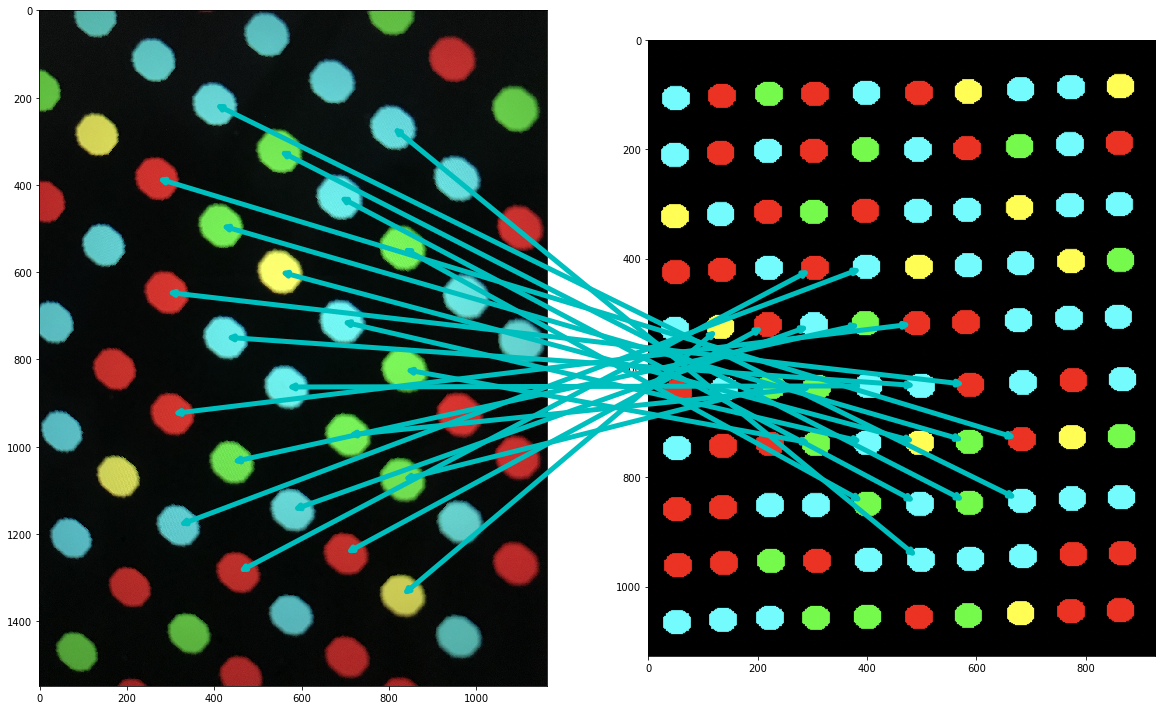
Pattern: RRGGBBYRBFound at (385, 269) in image 1Found at (731, 682) in image 2

Pattern: BYBGBBBGGFound at (427, 686) in image 1Found at (849, 496) in image 2

Pattern: GRBRBYBGBFound at (493, 415) in image 1Found at (735, 586) in image 2

Pattern: GRBBBYBGBFound at (544, 831) in image 1Found at (849, 400) in image 2

* Line connections are used to show this.



**Test-Cases:**

Our validation set consisted of more than 20 images of the self-created 10x10 color codes. These were created using the above-mentioned criteria. Images were taken from different sources like mobile camera, laptop screenshots, and OpenCV save method after random crops.

Image-set

|  |
| --- |
|  |

**Results:**

|  |
| --- |
| Inserting image... |

|  |
| --- |
|  |

**Run-time:**

Average runtime of the above program on tested images was 0.869866132736206 s

|  |  |
| --- | --- |
| **Sub-program** | **Time Complexity** |
| Segmentation | O(N^2) |
| Nearest Neighbors Extraction | O(N^2) |
| Pattern Formation | O(N) |
| Matching Same Points | O(N^2) |

**Limitations:**

We did not take into account the effect of wrong patterns formed at the edge circles, since the 3x3 pattern is not complete at that position. However, this can easily be corrected using a combination of Homography and RANSAC, which is beyond the requirements and scope of this project.

**Code:**

Upload .ipynb file to GitHub and attach link here!