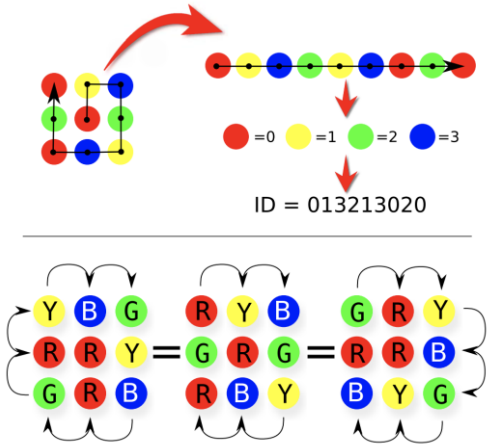
GARMENT-BASED MOTION CAPTURE

ABSTRACT:

The project is built on the findings in the paper ‘GaMoCap’ by Francesco Setti & co. We propose a new system for motion capture that consists of a 3x3 color-coded garment dress and cameras for image capturing. The high density of circular/irregular markers on the cloth gives an edge in matching similar features across various images. This makes the reconstruction of the scene easy and we can analyze the trajectory of motion of points from fixed camera angle images. The system works for RGB as well as grayscale images with high accuracies under controlled light conditions.

INTRODUCTION:

The basis of our motion reconstruction is a combination of 3x3 codes as shown in the diagram. An ID is assigned to each 3x3 pattern and the center of patterns are matched between two corresponding images.

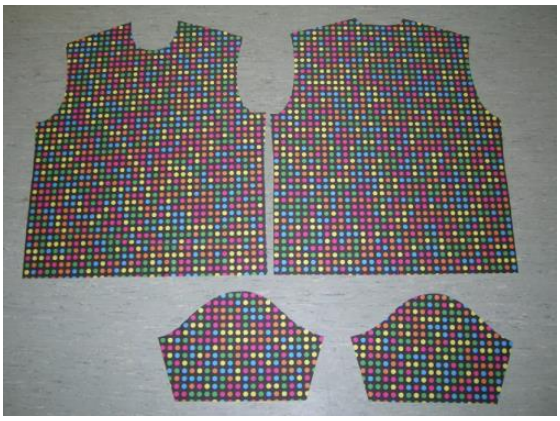


The maximum number of ID that can be created with a sequence of 9 elements and 4 colors is 4^9 = 262, 144. Not all the sequences are rotation invariant. Based on the equation shown below, a maximum of 32640 unique IDs can be formed using 4 colors that satisfy the requirements.

$$\frac{1}{d}s_d \quad \text{where} \quad s_d = q^d - \sum_{i<d, i|n} s_i.$$

q=4 (unique colors)
m=8 (total elements / spaces)
d=8 (no.of shifts)
n=9 (3x3 block)

These patterns can be generated to construct garments and thus simplify matching features on a human body, as opposed to texture less and simple clothes.



SOLUTION:

Our project is divided into four phases:

Phase 1: Generate the color-coded pattern with the required properties of uniqueness of pattern considering circular shift.

Phase 2: Test color detection in images and match patterns in aligned images.

Phase 3: Further the algorithm to work on any RGB image. Given two images as input, find all the same points/features in both images and highlight them.

Phase 4: Finetune the algorithm for grayscale images by defining grayscale levels.

Part-1: Automatic Pattern Generation

Objectives:

Generate unique 3x3 color-coded pattern for garments to simplify feature matching across images. Use 4 distinct colors.

Approach:

To automatically generate the color grid of required size, we have used following steps:

- Forming a 2D grid of required size.
- For creating unique grid, first we generate 3x3 grid of random values as a starting grid.
- Traversing the grid first horizontally to append random new 3 elements at the end column of each grid satisfying the property of **Circular Shift Invariance.**
- Then Traversing the grid vertically and appending 3 new elements at the end row of each grid satisfying the property of Circular Shift Invariance.
- In case the newly generated pattern or grid doesn't satisfy Circular Shift Invariance, we keep on generating a new random grid until it satisfies the given property.
- After having this grid, we used ImageDraw module draw.ellipse to form circles corresponding to newly generated image grids.
- Color of the Circle is chosen corresponding to the respective values in the given grid.

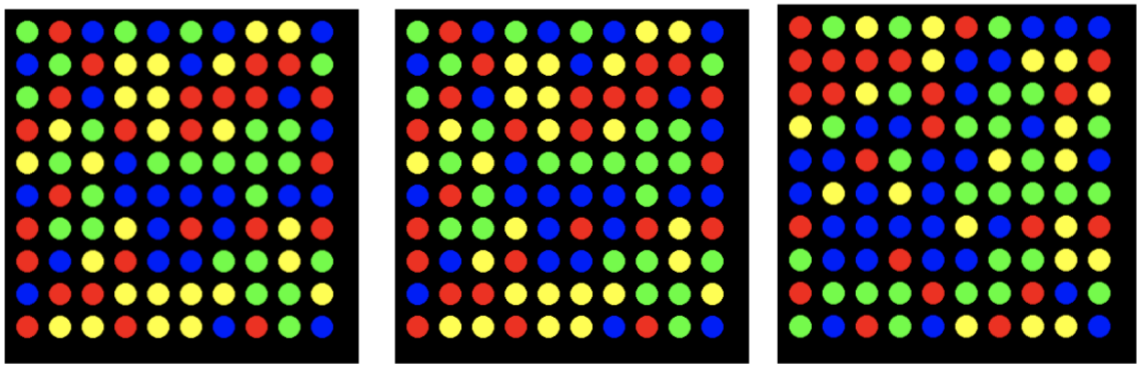
Results:

While our method can automatically generate more than 1000 patterns sufficient for an entire garment, our requirement was only to form a 10x10 grid.

```
my_patterns
```

```
array([[3, 2, 2, 2, 1, 0, 0, 0, 1, 0],
       [0, 3, 2, 2, 1, 0, 1, 2, 2, 1],
       [1, 0, 2, 2, 2, 3, 1, 0, 3, 1],
       [1, 2, 0, 1, 2, 1, 1, 0, 3, 3],
       [0, 1, 2, 0, 2, 1, 1, 1, 2, 3],
       [3, 0, 1, 1, 0, 0, 1, 0, 3, 0],
       [3, 2, 3, 1, 0, 2, 0, 1, 3, 1],
       [0, 1, 2, 2, 2, 2, 2, 3, 2, 3],
       [1, 1, 2, 3, 2, 1, 1, 1, 0, 0],
       [0, 3, 0, 0, 0, 0, 0, 2, 0, 3]], dtype=uint64)
```

Try 1,2,3: Unique 10x10 pattern-generator



Limitations:

The pattern generation gets tricky for very large sized images e.g. 1000x1000 grid. Because of the lack of append able combinations, the algorithm takes large runtime.

Part-2: Matching patterns across aligned images

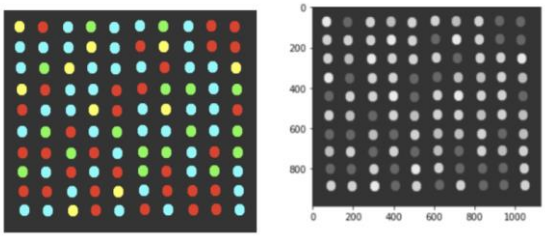
Objectives:

Match 3x3 patterns across aligned RGB and grayscale images. It is assumed that we are provided with the homography which can help us rotate and align the images.

Approach:

- **Color selection:**

  This is done initially for RGB Images and then for Gray Scale as well. Definite colors like Red, Yellow, Green and Blue are defined on all 3 channels, however for Gray Scale Images different shades over only one channel has been defined for distinguishing between different dots and between the background color/shade.



- **Circle Detection Algorithm:**

  For detection of dot any first pixel's color of the dot is detected and detection is halted until Black/background color is detected. For greater accuracy we average the first and last non-black pixel gives center's x-coordinate. In this way we have traversed the whole image and refined the result by filtering out the background colors from our output. After Dot detection we have generated an array of grids corresponding to the colors of the dots.

```
[['R', 'R', 'G', 'R', 'Blue', 'R', 'Y', 'Y', 'R', 'Y'],
 ['Y', 'R', 'Y', 'G', 'Y', 'Blue', 'Blue', 'Y', 'R', 'R'],
 ['Y', 'Y', 'Y', 'Y', 'G', 'Blue', 'Y', 'G', 'R', 'Blue'],
 ['Blue', 'Blue', 'Blue', 'G', 'G', 'Blue', 'Y', 'G', 'Y', 'Y'],
 ['R', 'Y', 'R', 'G', 'Y', 'R', 'R', 'R', 'R', 'G'],
 ['Blue', 'G', 'Y', 'Y', 'R', 'G', 'G', 'R', 'Y', 'Blue'],
 ['Y', 'Blue', 'G', 'G', 'Y', 'Blue', 'G', 'G', 'Blue', 'G'],
 ['G', 'Blue', 'Blue', 'G', 'G', 'G', 'R', 'Blue', 'Y', 'R'],
 ['Y', 'G', 'R', 'Y', 'Y', 'Y', 'R', 'R', 'R', 'Blue'],
 ['G', 'Blue', 'R', 'G', 'R', 'Y', 'Blue', 'Y', 'Blue', 'Y']]
```

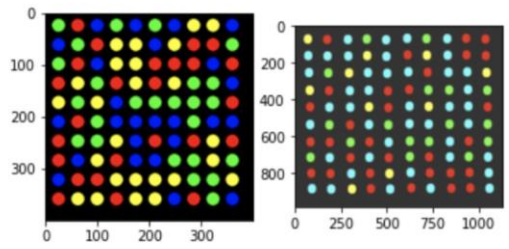
- **Circular Shift Invariance**

  By satisfying Circular Shift Invariance on the extracted grid, we have extracted out all possible patterns present in the given image.

```
array([[1, 3, 1, 1, 0, 2, 1, 1, 0],
       [1, 1, 2, 0, 1, 0, 2, 1, 3],
       [0, 2, 1, 1, 1, 1, 0, 1, 1],
       [1, 1, 1, 3, 1, 1, 1, 0, 2],
       [3, 1, 2, 0, 3, 1, 1, 1, 1],
       [0, 2, 1, 1, 1, 3, 1, 3, 1],
       [1, 1, 3, 3, 1, 1, 3, 0, 2],
       [3, 3, 3, 3, 0, 1, 1, 1, 1],
       [2, 1, 1, 0, 1, 3, 0, 1, 1],
       [0, 1, 0, 1, 1, 1, 3, 2, 1],
       [1, 0, 1, 1, 3, 1, 1, 0, 1],
       [1, 1, 3, 1, 3, 3, 1, 1, 0],
       [1, 3, 0, 3, 2, 3, 3, 1, 1],
       [3, 0, 1, 1, 2, 2, 3, 1, 3],
       [1, 1, 3, 1, 1, 2, 2, 3, 0],
```

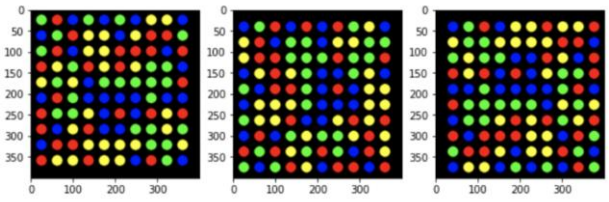
Results:

1) **Two different Images have been given with no Match:**



```
Total number of patterns matched are : 0
```

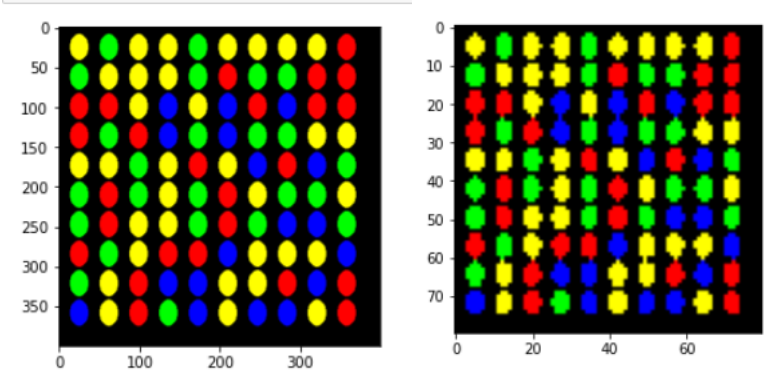
3) **Same Images have been given (One rotated version of previous Image):**



```
Total number of patterns matched are : 64
```

Distorted Image Handling:

For the objects other than round circular dots, our solution works fine too. As it is described above our solution does not dependent on specific shapes because we are using image Traversal approach. Following are the results for distorted dots and their extracted.



Same results for both:

```
([['Y', 'G', 'Y', 'Y', 'G', 'Y', 'Y', 'Y', 'Y', 'R'],
  ['G', 'Y', 'Y', 'Y', 'G', 'R', 'G', 'G', 'R', 'R'],
  ['R', 'G', 'Y', 'Blue', 'Y', 'Blue', 'G', 'Blue', 'Y', 'R'],
  ['Y', 'Y', 'G', 'Y', 'R', 'Y', 'Blue', 'R', 'Blue', 'G'],
  ['G', 'R', 'G', 'Y', 'G', 'R', 'Y', 'G', 'G', 'Y'],
  ['G', 'R', 'Y', 'Y', 'G', 'R', 'G', 'Blue', 'Blue', 'G'],
  ['R', 'Y', 'Y', 'Blue', 'R', 'Blue', 'Y', 'Y', 'Blue', 'Blue'],
  ['Blue', 'Y', 'R', 'G', 'Blue', 'Y', 'Blue', 'Blue', 'Y', 'R']],
```

Advantages:

Fast and efficient method for pattern detection.

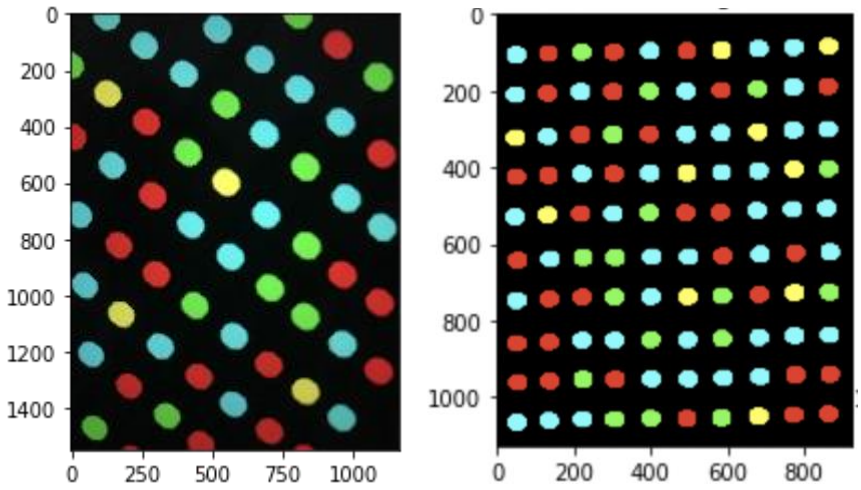
Limitations:

The Image traversal approach can work only on specific angles (0,90,180,360). So, it cannot work for perspective as well as rotated Image which are handled by another approach explained at a later stage. Assumption that homography is known and image can be rotated and aligned to form a rectangular grid.

Part-3: Matching patterns across all RGB images

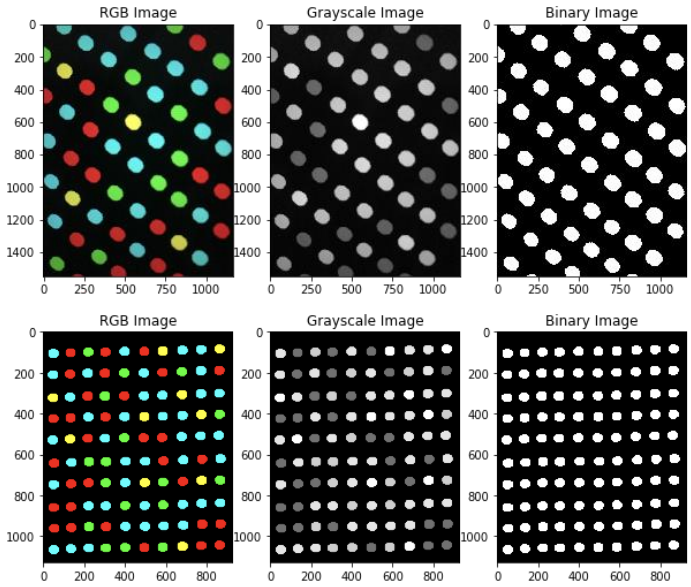
Objectives:

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

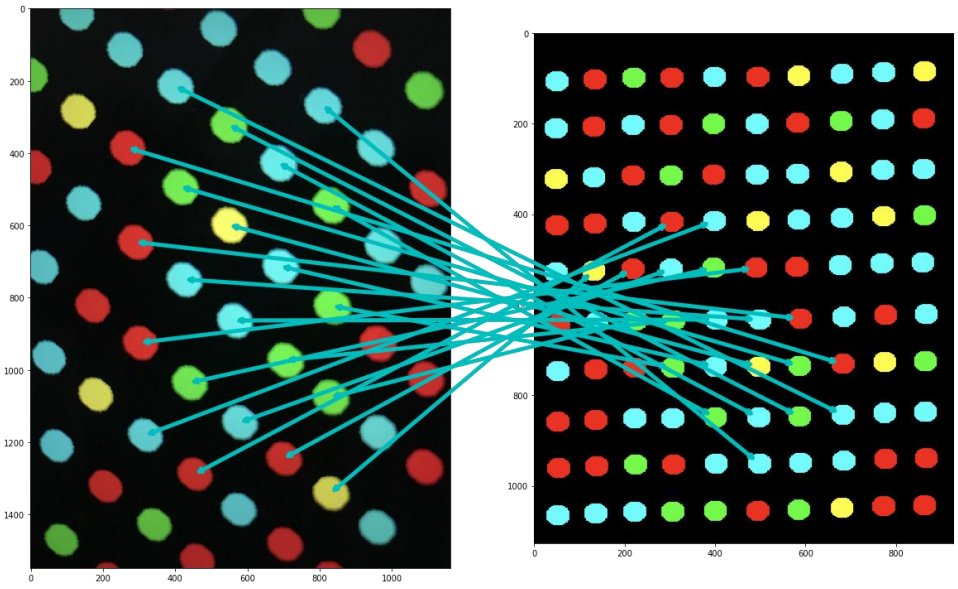
  {(26, 129): ['320130330', '302013033', '330201303', '333020130', '303302013', '330330201', '313033020', '301303302'], (0, 383): ['033103323', '033310332', '023331033', '032333103', '033233310', '003323331', '010332333', '031033233'], ….}

- 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
  …..

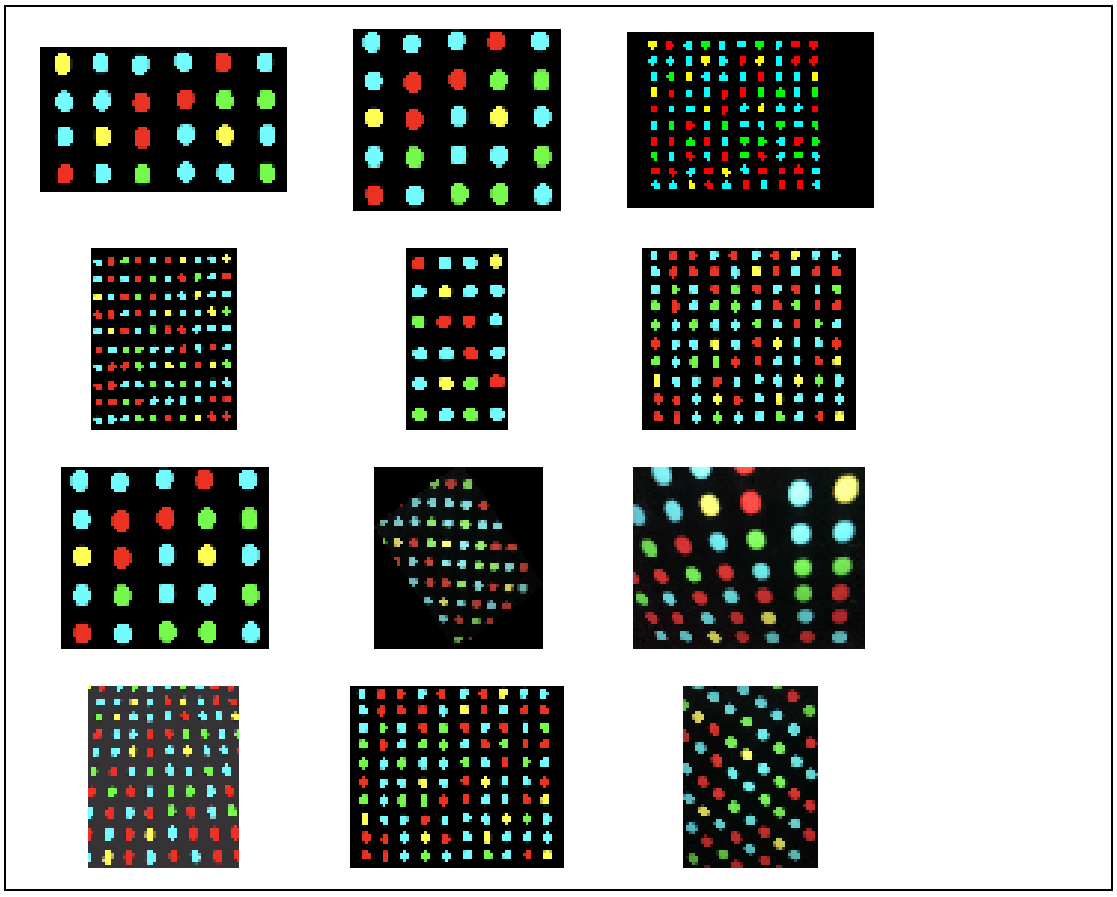
- Line connections are used to show this.



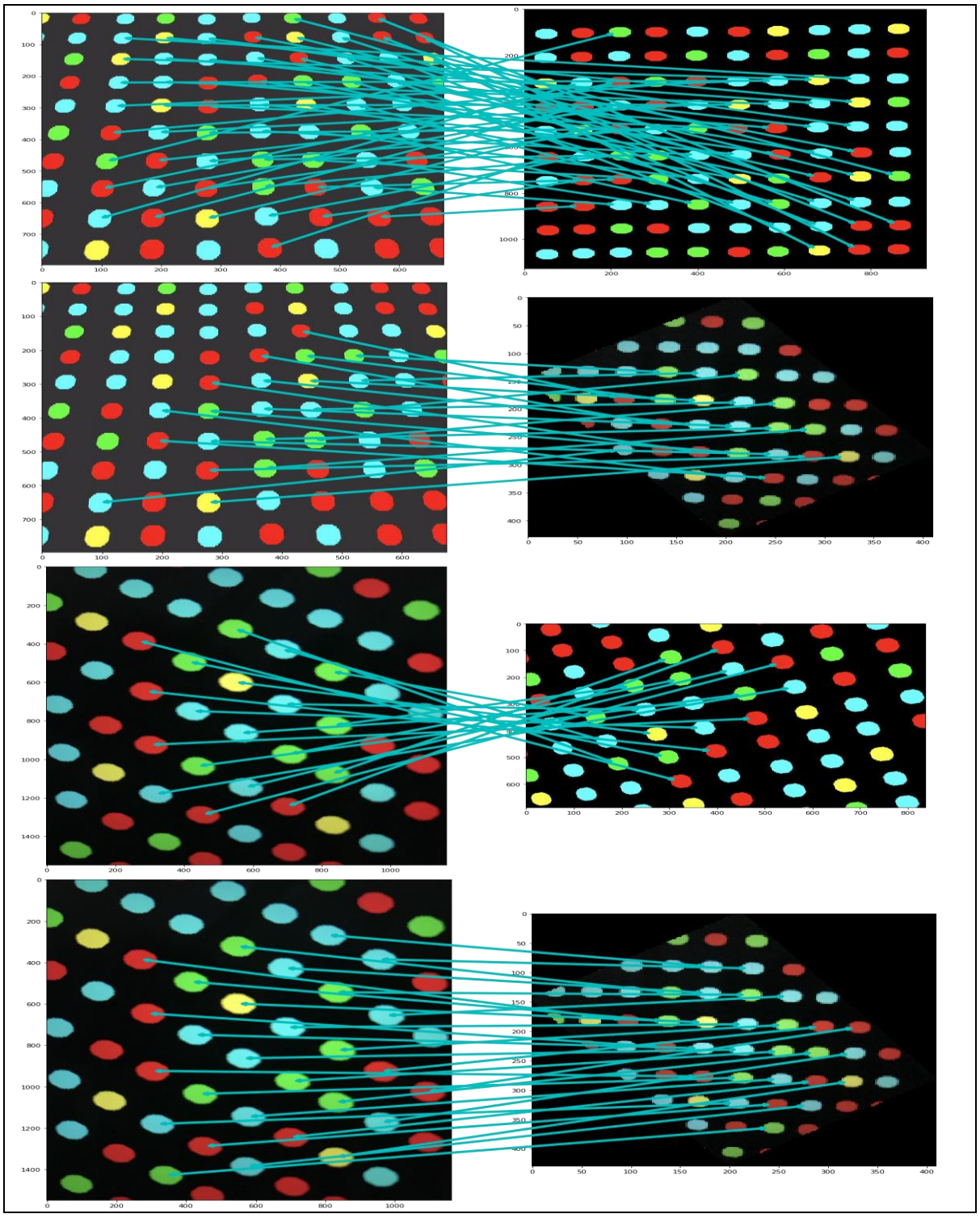
Test-Cases:

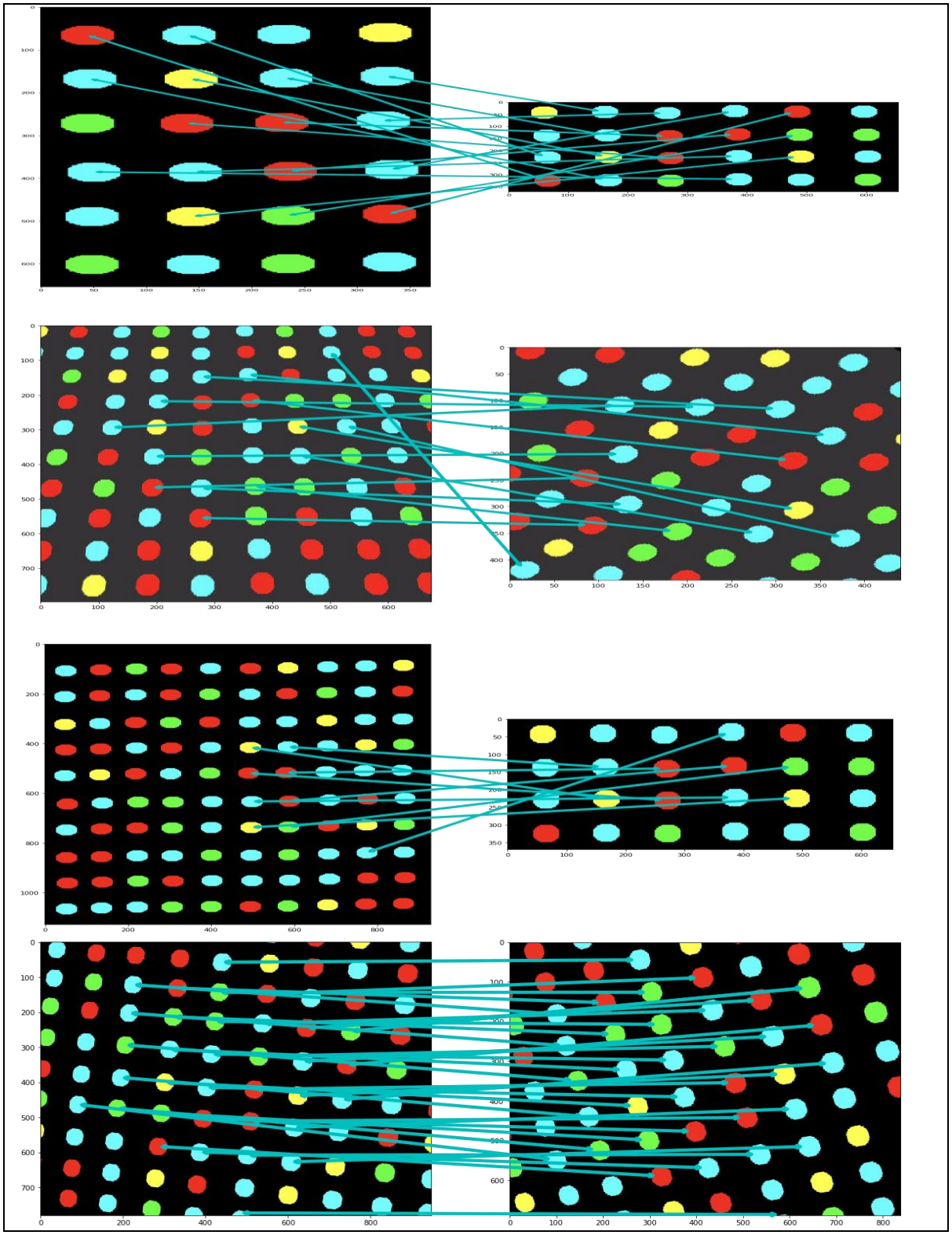
Our validation set consisted of more than 20 images of the self-created 10x10 color codes. These were created using the above-mentioned criteria. Images were taken from different sources like mobile camera, laptop screenshots, and OpenCV save method after random crops.

Image-set



Results:





Run-time:

Average runtime of the above program on tested images was 0.869866 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.

Part-3: Matching patterns across Grayscale images

Objectives:

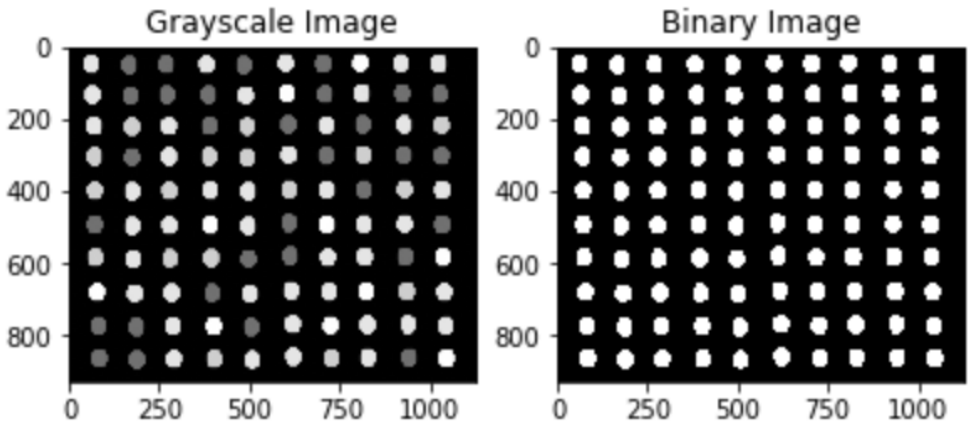
Match 3x3 color-coded patterns across two grayscale images. This is important considering the cost benefits of using monochrome cameras.

Approach:

The approach in grayscale images is the same as used for RGB images. Same steps of component extraction, nearest neighbor calculation, pattern orientation, pattern coding, and matching features across images are used. The only difference is in the color identification. Four customized gray levels are used for this purpose and are identified using the same mean-squared approach. These levels on a 0-255 scale are:

A = 100, B = 240, C = 170, D = 210, E = 35 (background)

ID representation: {0:'A', 1:'B', 2:'C', 3:'D', 4:'E'}



Limitations:

Grayscale color identification is a bit tricky because of the closeness of gray levels. Thus, some mobile captured images were causing color identification problem. However, since our goal is to capture garment images under controlled light conditions, and in such case the program works perfectly.

Algorithm flow:

{centroid:color}

{(48, 597): 'D', (48, 803): 'B', (48, 915): 'D', (50, 701): 'A', (50, 1019): 'D', (52, 61): 'D', (52, 267): 'A', (52, 379): 'D', (54, 165): 'A', (54, 483): 'A', …}

{centroid:{nearest_8:color}

{(48, 597): {(50, 701): 'A', (54, 483): 'A', (132, 601): 'B', (134, 705): 'A', (138, 487): 'D', (218, 605): 'A', (220, 709): 'D', (224, 491): 'C'},

(48, 803): {(48, 597): 'D', (48, 915): 'D', (50, 701): 'A', (132, 807): 'D', (132, 919): 'A', (134, 705): 'A', (218, 811): 'A', (220, 709): 'D'}, …}

{centroid:pattern}

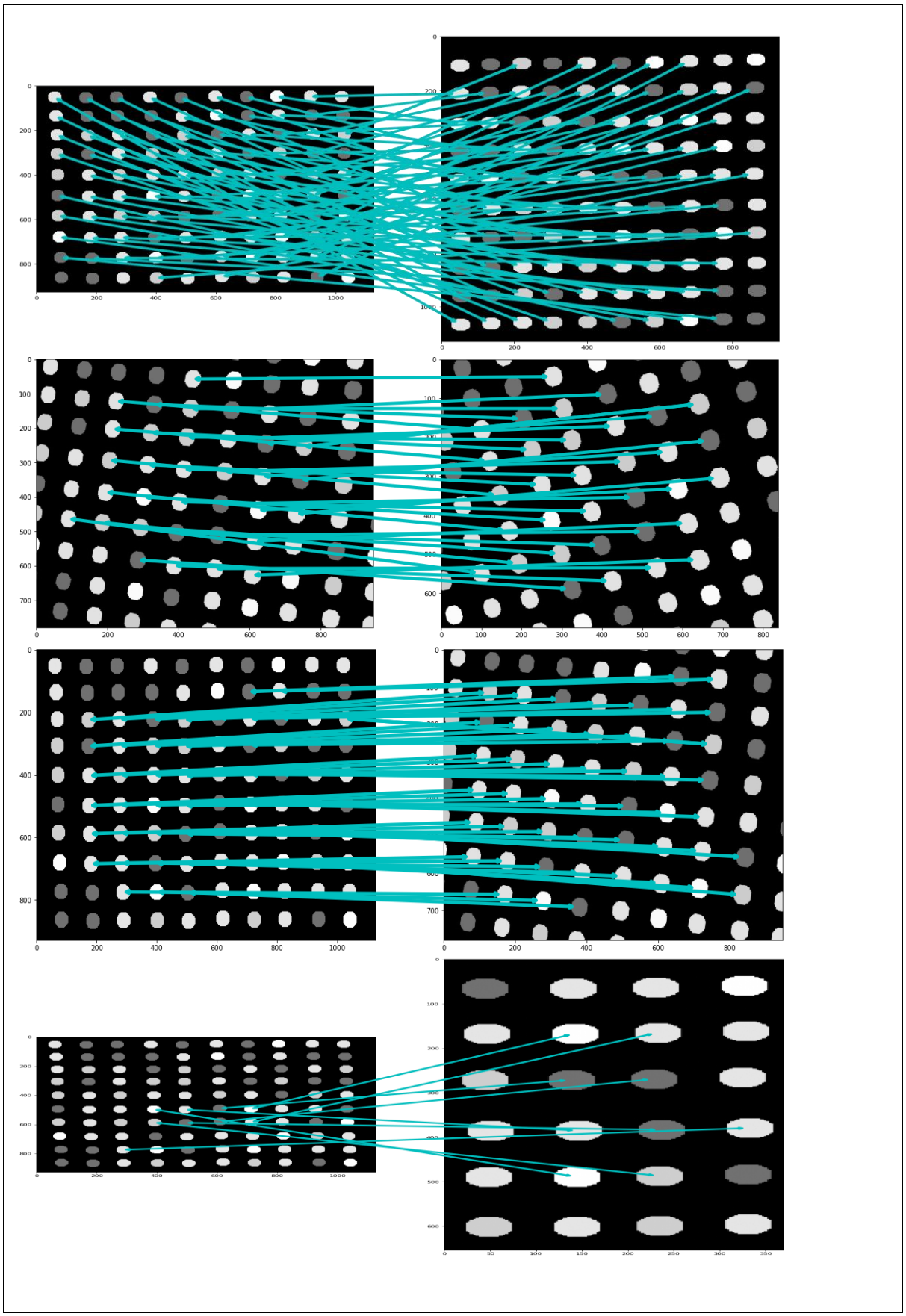
{(48, 597): 'DADCBAADA', (48, 803): 'BDAADDAAD', (48, 915): 'DBDAADACD', (50, 701): 'ADBAADDAB', (50, 1019): 'DBDDADACA', (52, 61): 'DDDCACAAA', (52, 267): 'ADAACADAD', (52, 379): 'DAADAADCA', (54, 165): 'ADDDACADA', (54, 483): 'ADAADCBAD', …}

{centroid:pattern_IDs}

{(48, 597): '303210030', (48, 803): '130033003', (48, 915): '313003023', (50, 701): '031003301', (50, 1019): '313303020', (52, 61): '333202000', (52, 267): '030020303', (52, 379): '300300320', (54, 165): '033302030', (54, 483): '030032103', (132, 601): '103203003', (132, 807): '300303031', (132, 919): '013032033', …}

This is followed by rotation variants code and same points matching just like in RGB images. The results are shown below which are comparable with the accuracy achieved in RGB images.

Results:



CONCLUSION:

Feature matching was done in real-time with relatively high accuracy with the help of markers. The high density of such markers makes it possible to easily reconstruct motion and plot the trajectory of moving points across images.

FUTURE APPROACH AND POSSIBILITIES:

Print patterns on clothes and try detection with cameras on real garments. Segment the required pattern portion from the camera view and match all patterns/features in different images. This can be used to reconstruct motion and has many applications like model clothes simulation when it comes to online shopping.

CODE FILES:

Shared on Google Colab.

REFERENCES:

https://link.springer.com/article/10.1007/s00138-015-0701-2

https://graphics.tu-bs.de/upload/people/magnor/publications/eg05.pdf

https://www.researchgate.net/publication/47863940_Garment_Motion_Capture_Using_Color-Coded_Patterns