Image Processing Project

1 Overview

In this project, we are requested to create a function that will take as input an image that represents every time a coloured matrix in different projections and orientations. More precisely, each time the images contain a matrix that is consisted of 16 squares where each square is randomly coloured with one individual colour. The function then will have to correctly identify the colour of each square and display its output in a 4x4 matrix that is reflecting the given image. The colours that are used in the images are white, blue, green, red, and yellow: and for better understanding, the output matrix is displaying the identified colour with its first letter. The set of images that the code has been written for is the 'images2' set, which as I mentioned above it contains images in various orientation and projections and images with noise.

2 Strategy

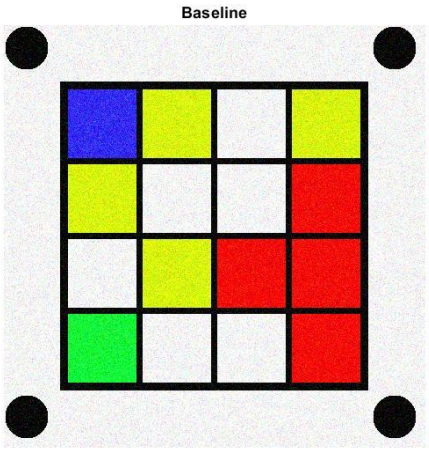
First, the problem was split into two parts. The first part was about translating all the images in our set, images that are differently rotated and projected into a standard orientation through finding common points with our baseline image ('org_1'). The second part was about identifying the individual colour of the squares: this was achieved by cropping a small window area around the centroid of each square and reading its pixel colour values. Finally, the output was displayed in a 4x4 matrix as was mentioned earlier.

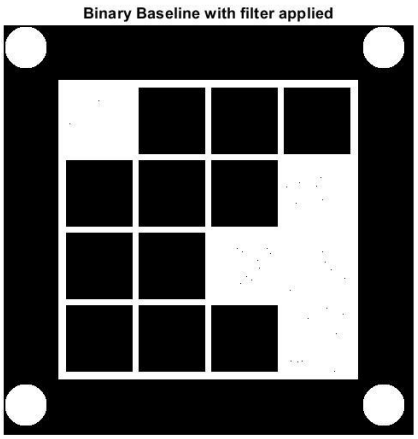
2.1 1st Part: Image Transformation

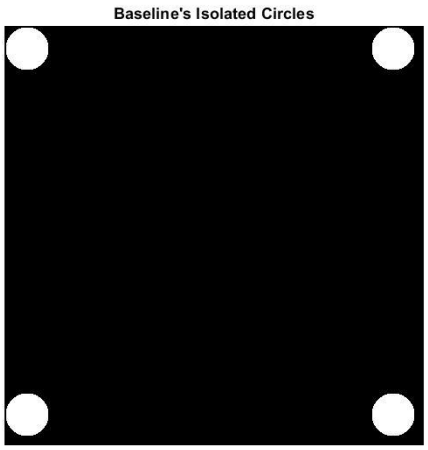
To begin with, to correct the orientation of each input image, "org_1" image was loaded first in the function. This image was in the correct orientation and projection, thus served as the baseline for all the transformations. As I mentioned above, the main strategy was to find common points through our baseline image and use them as reference points to correct the input images. The common points that were chosen were the centroid coordinates of the 4 corner circles in the image. After all the points were obtained, the transformation took place through the fitgeotrans function, had its size corrected through the imref2d function and finally got displayed by the imwarp function.
More precisely, some image pre-processing needed to be done before the object identification and the circle isolation processes. Firstly, the baseline image was converted into binary. Next, an erosion followed by a dilation was applied to denoise the image. The pre-processing was used to remove each small mass of unwanted area, letting only the significant objects in the image. Now that the image is clean, bwconncomp and regionprops built-in functions, from the image processing toolbox, were used to identify the objects of the image and calculate their properties. After the properties were calculated, it was noticed that the circles could be easily distinguished from the other objects,

by their area mass. This meaningful insight then was used within a loop removing the maximum area and eventually displaying an image that has the circles isolated. Next, bwlabel function was used in the new, displayed image. This built-in function labelled each blob found in the image, in our case circles, and afterwards, with the help of regionprops, the centroid coordinates of each of these blobs were found. Finally, once the points were obtained, they were recorded in an array called FixedPoints.

Below, for better understanding, the main tasks mentioned above were demonstrated visually.

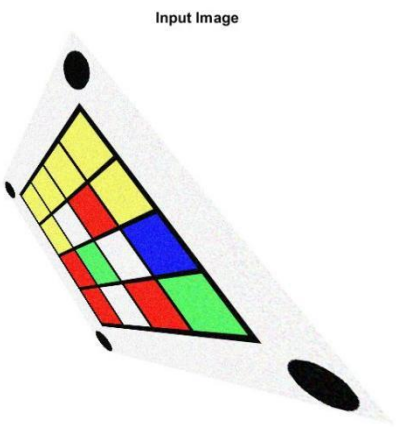


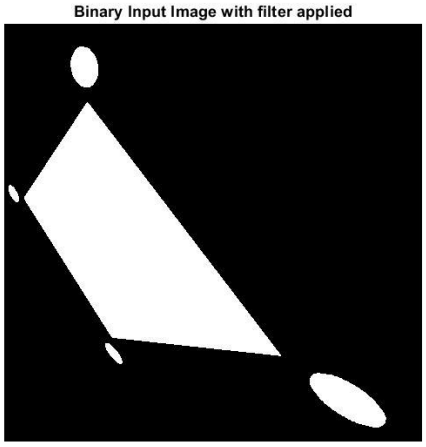


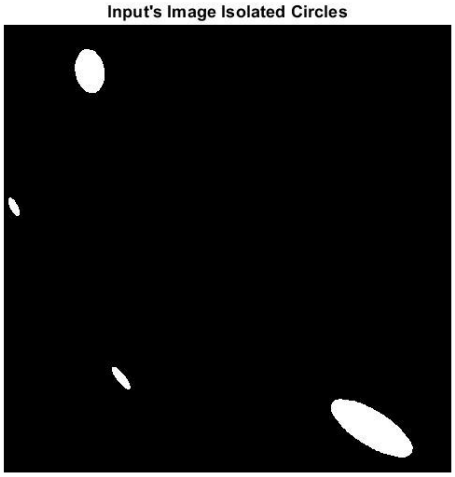


The exact same process was followed for the input image, however stronger denoising filters had to be applied as the images were getting noisier, thus harder to be cleaned. The new points found in the input image were now recorded in an array called MovingPoints.

Below, “proj_2” image was randomly selected, as an input image to demonstrate the process visually.

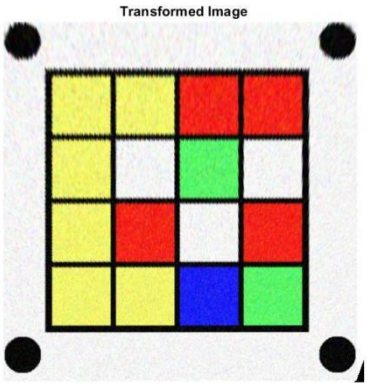






Now that all the points have been obtained and been recorded in their corresponding arrays, the built-in function fitgeotrans was used to perform the geometric transformation. This function accepts three arguments. Specifically, the first two inputs are referring to the two sets of points that we already have obtained, the ones that have been recorded as our reference points (FixedPoints) and the ones that we want to get fixed (MovingPoints). It requires these inputs so it can find matches between the two set of points, and then be able to translate the MovinPoints through the FixedPoints. The last one refers to the type of transformation we want to perform. Since we want to capture all the various rotations and projections that our set of images contains, the ‘projected’ one was selected. Finally, the imref2d function was used to resize our image with reference to the baseline image, and then imwarp function transformed the image according to the geometric transformation that was applied above.

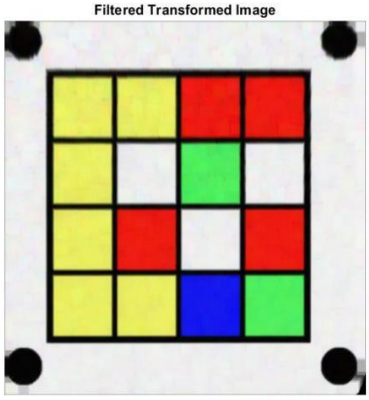
Here, the transformation of the input image, that was used above, can be seen in the below image.



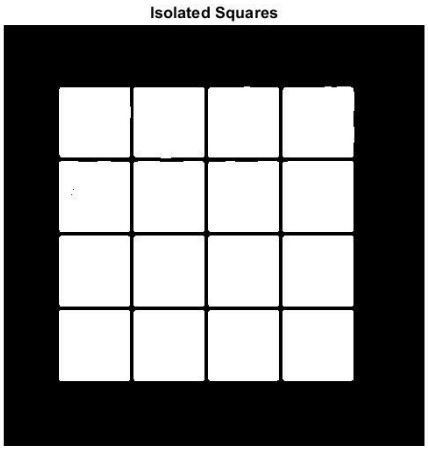
2.2 2nd Part: Colour Recognition

Now that the image is in the correct rotation and projection it is time for the algorithm to move on to the colour recognition part. As I mentioned above, the main strategy was to calculate the colour pixel values around the centroid coordinates of each square and then translate them into their corresponding colour, in the output matrix. This was achieved, by converting the image into the Lab colour space and then finding meaningful thresholds, that could easily classify each colour.
More precisely, before converting the image into the Lab colour space, image pre-processing needed to be done to denoise the image, as also to help with the edge detection. Firstly, a Gaussian filter was applied to remove the big noise in the image. Next, an erosion followed by a dilation was applied to clear the blurring that Gaussian filter created. The next process that took place was the edge detection. Specifically, the clean image was first converted into grey and then the edges were detected using the 'canny' method. Finally, the edges were sharpened using dilation with a manually fixed structured element.

Here, the transformed image with filter applied can be seen in the below image.



Now, that the noise removal was done, and the edges were detected, bwconncomp and regionprops functions were used, again, to identify the objects of the image and calculate their properties. In contrast to the first part (Image Transformation) where the circles needed to be isolated, meaningful insights were now required to be found to isolate the squares. It was noticed that the area mass of an object, was again the most useful stat that could help to distinguish all the squares from the other objects of the image. Finally, following the exact same process that was performed for the circle isolation, the squares were isolated and afterwards their centroid coordinates were calculated. The isolated squares of the transformed image can be seen in the image below.



After having obtained the centroid coordinates of each square, the algorithm is ready to continue the colour identification task. Here a problem was occurred with the built-in function bwlabel, as it was labelling each square of the image in an arbitrary order, as a result creating problems to the next processes. Considering that all the images in our set, are being transformed regarding the same baseline, they have almost the same rotation and projection, thus we can assume that the centroid coordinates of the matching squares are almost in the same position. This hypothesis was approved correct after inspecting the values that each input image was returning. As a result, the problem was dealt with by initialising an array and then manually appending the centroid coordinates of the squares, in the correct order, rounded.

Now that the problem is being solved, it is finally time to proceed to colour recognition. Firstly, an empty cell array of size 4x4 was created, where all the values that will be returned during the colour recognition task, will be appended there reforming it eventually into a matrix that reflects the input image. Then each colour, that is being used in the set of images, was translated into its fixed values, in the Lab colour space, and recorded in its matching array. These values were found and appropriately fixed, after inspecting the minimum and maximum values that each coloured square return in the Lab colour space. Initialising these five arrays, was a method to create thresholds that will help the algorithm to distinguish the colours. Next, a loop was created ranging through the points in the array, that the centroid coordinates were recorded. The main steps of the loop were, firstly to create a small window around the centroid coordinates of each square, next to calculate the mean L, a, b values in each window and record them to an array, then, using the Euclidean

distance, calculate which one of the five arrays that represent the colours has the minimum distance to the array, that holds the mean L, a, b values of the colour that is being processed, and finally append the output to the empty cell array, that was initialised outside the loop. MeanIntensity stat, that regionprops function provides, could be used as an alternative approach instead of calculating the mean L, a, b values of each square, if and only if, the bwlabel function could label the blobs in the correct order.

3 Live Images

To begin with, if we try to recognise live images, a baseline image which has a standard orientation is required to be taken again. Looking further, in the set of live images, it is noticed that most of them have inconsistent lighting with either patch of shadows or patches with various intensity range, making them nonuniform illuminated, thus the baseline image needs also an ideal lighting.
A possible solution to countering the images that have inconsistent lighting could be a histogram normalisation. More precisely, the histogram for both (input image and baseline image) would be found, followed by the intensity range of the input image that would have to be translated into the intensity range of the baseline image.
As for the blurred images that the set contains, a strong edge detector method could be applied like canny, which is the least likely to be affected by noise. Afterwards, SIFT matching could be selected to correct the orientation of the input image. Finally, after all these processes are considered finished, the colour recognition process would be the same as the methodology performed above, eventually outputting the results into a matrix.

4 Summary

To sum up, in this project we were requested to create a function that accepts a colour squared image,then outputs a matrix of size 4x4 that reflects the image with its elements to be the initial letters of the corresponding colour.
Due to the variety of the orientations and projections that each image has in the set of images, the processes had to split into two parts. The first part was related to the image transformation, while the second part was about colour recognition. More specifically, in the first part, an image that was in the correct orientation and projection served as the baseline. Then, each image that had different orientations and projection was translated into the standard shape by finding common points with the baseline. On the other hand, the second part was achieved by firstly converting the image into the LAB colour space. Afterwards, by using suitable thresholding, each pixel around the centroid coordinates of the square was translated into the corresponding colour.
According to the results, in the appendix, the accuracy of the function on reflecting the image into a matrix seems to be one hundred per cent. Improvements that could be made, are including the time cost of the function. More specifically, more effective, and more suitable denoising techniques could be found to reduce the number of the pre-processing tasks, that the algorithm follows. Also, an improvement could be made by finding a way to automatically fix the array that represents the centroid coordinates of each square in the image, instead of initializing it manually.

5 Appendix

5.1 Code

```
function result = colourMatrix(filename)

                                % Image Transformation %

baseline=imread('org_1.png'); % Reads the image, Here the org_1 image was chosen as our baseline
image.
%figure,imshow(baseline),title('Baseline')
baselinebw  =~ im2bw(baseline,0.5); % Converts the image to binary (black and white).
erobaselinebw = imerode(baselinebw,ones(7)); % Erosion performed for denoise purpose.
dilbaselinebw = imdilate(erobaselinebw,ones(7)); % Dilation was used to sharp the eroded image.
%figure,imshow(dilbaselinebw),title('Binary Baseline with filter applied').

                                % Find Circle Regions %

CC = bwconncomp(dilbaselinebw); % Finds the connected components of the image.
z = regionprops(dilbaselinebw,'Area','Centroid'); % Calculates stats and find useful insights (that will
be used as thresholds to help the circle recognition).
Areas = [z.Area]; % Making an array saving the area mass of each object.
%Areas
                                % Isolate Circles %

isolatedCircles=zeros(CC.ImageSize); % Makes a zero array, with equal size as the image.
for p=1:CC.NumObjects  %Loops through each object, identified from bwconncomp.
   if z(p).Area < max(Areas) % Condition statement that returns only objects that have smaller area
mass than the maximum area.
      isolatedCircles(CC.PixelIdxList{p}) = 1; %Set the image.
   end

end
%figure,imshow(isolatedCircles,[0 1]) %Display with fixed intensity range.
%title("Baseline's Isolated Circles")
labeledCircles = bwlabel(isolatedCircles, 8); %Labels each blob in the image, in our case, circles.
CircleMeasurements = regionprops(labeledCircles, 'all'); %Calculates stats of the circles.
allCircleCentroids = [CircleMeasurements.Centroid]; %Saving the four centroids of the four circles, in
an array.
FixedPoints=reshape(allCircleCentroids,2,[]); %Sets the centroids as moving points.
tFixed = transpose(FixedPoints); %Transposes the matrix so it will have the form [xi yi; xi+1 yi+1; and
so on].

                                % Load in the input image to be transformed %

inputImg=imread(filename) ;
%figure,imshow(inputImg),title('Input Image')
```

```
inputBW =~ im2bw(inputImg,0.5);
inputBWfill=imfill(inputBW,'holes'); % Denoises the image, filling noise holes.
inputMedfilt = medfilt2(inputBWfill); % Removing salt and pepper noise using the non-linear median
filter.
%figure,imshow(inputMedfilt), title('Binary Input Image with filter applied')

                                        % Isolate Circles %

CC2 = bwconncomp(inputMedfilt);
s = regionprops(inputMedfilt,'Area','Centroid');
Areas2 = [s.Area];
isolatedCircles2 = zeros(CC2.ImageSize);
for p = 1:CC2.NumObjects
   if s(p).Area < max(Areas2)
      isolatedCircles2(CC2.PixelIdxList{p}) = 1;
   end
end
%figure,imshow(isolatedCircles2,[0,1]), title("Input's Image Isolated Circles")

labeledCircles2 = bwlabel(isolatedCircles2, 8);
CircleMeasurements2 = regionprops(labeledCircles2,'Area','Centroid');
allCircleCentroids2 = [CircleMeasurements2.Centroid];
MovingPoints=reshape(allCircleCentroids2,2,[]); %Set Centroids as moving points.
tMoving = transpose(MovingPoints);

                                % Applying the geometric transformation. %

tform=fitgeotrans(tMoving,tFixed, 'Projective');

                                        % Size Correction %

R=imref2d(size(baseline)); %Resizes the transformed image, with reference the baseline.
transformedImg = imwarp(inputImg,tform,'OutputView',R);
%figure,imshow(transformedImg),title('Transformed Image')

                                        % Colour Recognition %

h = fspecial('Gaussian',[3 3],2); %Creates a Gaussian filter.
blurred = imfilter(transformedImg,h); %Denoises the image using the Gaussian filter.
%figure,imshow(blurred)
se = strel('square',11); %Creates a structure element, that will be used later.
erosion = imerode(blurred,se); %Erosion performed, using the structure element to clear the blurred
image.
%figure,imshow(erosion)
dilation = imdilate(erosion,se); %Dilation followed for last noise removal.
figure,imshow(dilation),title('Filtered Transformed Image')
C = makecform('srgb2lab'); %Converts the image into the Lab color space.
transformedImgLab = applycform(dilation, C);
%figure,imshow(transformedImgLab)
Gray = im2gray(transformedImgLab ); %Converts the image to the grayscale.
%figure,imshow(binary)
```

```
edges = edge(Gray,'canny'); %Finds the most important edges, using the "canny" method.
%figure,imshow(edges)
se1 = strel('square',3); %Creates a new structure element
sharpened = imcomplement(imdilate(edges,se1)); %Dilation performed, sharpening the edges. Also
imcomplement used, reversing the black and white colours.
%figure,imshow(sharpened)

                                % Calculating stats %


 CC = bwconncomp(sharpened);
 squareMeasurements = regionprops(sharpened,'Area');
 Areas3 = [squareMeasurements.Area];
 %Areas3


                                % Isolating the Squares %

isolatedSquares=zeros(CC.ImageSize);
for p=1:CC.NumObjects  %loop through each image
   if squareMeasurements(p).Area < 6500 && squareMeasurements(p).Area > 4000
      isolatedSquares(CC.PixelIdxList{p}) = 1; %set the image
   end

end

%figure,imshow(isolatedsquares,[0 1])
cleaned = imdilate(isolatedSquares,ones(7)); %Erosion is taking place to fill the last gaps.
%figure,imshow(cleaned), title('Isolated Squares')

                % Investigating the Centroid Coordinates of the isolated squares %

labeledSquareImg = bwlabel(cleaned);
squareMeasurements = regionprops(labeledSquareImg, 'Centroid','PixelIdxList');
allsquareCentroids = [squareMeasurements.Centroid];
%Centroids2=reshape(squareMeasurements,2,[]);
%tr = transpose(Centroids2)

% Manually recording the Centroid Coordinates of each square in the images.
centroids =[ 113 113; 113 198 ; 113 283; 113 368; 198 113 ; 198 198 ; 198 283; 198 368; 283 113 ;
283 198; 283 283; 283 368; 368 113 ; 368 198; 368 283 ; 368 368];


                % Colours Translation into their L,a,b values %

white = [245,127,128];
blue = [81,145,61];
yellow = [231,114,214];
green = [227,74,204];
red = [133,190,192];
```

```
colors = cell(4);

for i = 1:16 %Loops through each square

   xmin = centroids(i); %x value of the centroid
   ymin = centroids(i,2); %y value of the centroid
   I2 = imcrop(transformedImgLab,[xmin ymin 15 15]); %Crops a small window around the centroid
coordinates of each square with width,height = 15
   L_Channel = I2(:,:,1); %L values
   a_Channel = I2(:,:,2); %a values
   b_Channel = I2(:,:,3); %b values
   meanL = mean(L_Channel); %Calculates the mean value
   meanA = mean(a_Channel);
   meanB = mean(b_Channel);
   thisColor = [meanL(i),meanA(i), meanB(i)]; %Records the mean Lab values, found above in an
array.
   distanceToWhite = sqrt((thisColor(1) - white(1)) .^ 2 + (thisColor(2) - white(2)) .^ 2 + (thisColor(3) -
white(3)) .^ 2); %Calculates the Euclidean distance between the fixed colours and the colour that is
being processed at a time.
   distanceToYellow = sqrt((thisColor(1) - yellow(1)) .^ 2 + (thisColor(2) - yellow(2)) .^ 2+ (thisColor(3)
- yellow(3)) .^ 2);
   distanceToRed = sqrt((thisColor(1) - red(1)) .^ 2 + (thisColor(2) - red(2)) .^ 2 + (thisColor(3) - red(3))
.^ 2 );
   distanceToBlue = sqrt((thisColor(1) - blue(1)) .^ 2 + (thisColor(2) - blue(2)) .^ 2 + (thisColor(3) -
blue(3)) .^ 2 );
   distanceToGreen = sqrt((thisColor(1) - green(1)) .^ 2 + (thisColor(2) - green(2)) .^ 2 + (thisColor(3) -
green(3)) .^ 2);
   distances = [distanceToWhite,distanceToYellow,distanceToRed,distanceToBlue,distanceToGreen];
%Records all the distances to an array.
   if distanceToWhite == min(distances) %Conditional statement that checks which fixed colour has
the minimum distance to the colour that is being processed at that time. Then it appends the
matching colour into the cell array and so on.
      colors(i) = cellstr('W');
   elseif distanceToYellow == min(distances)
      colors(i) = cellstr('Y');
   elseif distanceToRed == min(distances)
      colors(i) = cellstr('R');
   elseif distanceToBlue == min(distances)
      colors(i) = cellstr('B');
   elseif distanceToGreen== min(distances)
      colors(i) = cellstr('G');
   end
end

Results = reshape(colors,[],4);
result = Results
end
```

5.2 Results

Noise_1

```
Results =

  4×4 cell array

    {'B'}    {'Y'}    {'Y'}    {'B'}
    {'W'}    {'R'}    {'Y'}    {'G'}
    {'R'}    {'Y'}    {'Y'}    {'B'}
    {'G'}    {'Y'}    {'W'}    {'R'}
```

Noise_2

```
Results =

  4×4 cell array

    {'Y'}    {'B'}    {'R'}    {'G'}
    {'G'}    {'G'}    {'W'}    {'Y'}
    {'G'}    {'B'}    {'B'}    {'W'}
    {'R'}    {'Y'}    {'B'}    {'Y'}
```

Noise_3

```
Results =

  4×4 cell array

    {'R'}    {'R'}    {'R'}    {'Y'}
    {'B'}    {'B'}    {'Y'}    {'W'}
    {'G'}    {'B'}    {'Y'}    {'B'}
    {'R'}    {'B'}    {'W'}    {'W'}
```

Noise_4

```
Results =

  4×4 cell array

    {'Y'}    {'G'}    {'Y'}    {'W'}
    {'R'}    {'W'}    {'W'}    {'W'}
    {'R'}    {'W'}    {'G'}    {'G'}
    {'R'}    {'G'}    {'B'}    {'G'}
```

Noise_5

```
Results =

  4×4 cell array

    {'R'}    {'Y'}    {'R'}    {'Y'}
    {'B'}    {'G'}    {'Y'}    {'Y'}
    {'W'}    {'Y'}    {'B'}    {'G'}
    {'R'}    {'Y'}    {'B'}    {'Y'}
```

Org_2

```
Results =

  4×4 cell array

    {'W'}    {'G'}    {'Y'}    {'G'}
    {'W'}    {'R'}    {'B'}    {'W'}
    {'B'}    {'B'}    {'W'}    {'W'}
    {'G'}    {'Y'}    {'G'}    {'W'}
```

Org_3

```
Results =

  4×4 cell array

    {'G'}    {'R'}    {'Y'}    {'G'}
    {'Y'}    {'B'}    {'G'}    {'G'}
    {'B'}    {'R'}    {'R'}    {'W'}
    {'G'}    {'Y'}    {'W'}    {'Y'}
```

Org_4

```
Results =

  4×4 cell array

    {'W'}    {'W'}    {'B'}    {'Y'}
    {'G'}    {'G'}    {'B'}    {'R'}
    {'W'}    {'W'}    {'G'}    {'Y'}
    {'G'}    {'W'}    {'Y'}    {'R'}
```

Org_5

```
Results =

  4×4 cell array

    {'R'}    {'Y'}    {'G'}    {'Y'}
    {'W'}    {'G'}    {'G'}    {'G'}
    {'Y'}    {'W'}    {'G'}    {'R'}
    {'G'}    {'G'}    {'Y'}    {'W'}
```

Proj1

```
Results =

  4×4 cell array

    {'W'}    {'W'}    {'B'}    {'Y'}
    {'G'}    {'G'}    {'Y'}    {'W'}
    {'Y'}    {'Y'}    {'Y'}    {'B'}
    {'Y'}    {'G'}    {'R'}    {'Y'}
```

Proj2

```
Results =

  4×4 cell array

    {'Y'}    {'Y'}    {'R'}    {'R'}
    {'Y'}    {'W'}    {'G'}    {'W'}
    {'Y'}    {'R'}    {'W'}    {'R'}
    {'Y'}    {'Y'}    {'B'}    {'G'}
```

Proj3

```
Results =

  4×4 cell array

    {'W'}    {'G'}    {'W'}    {'B'}
    {'Y'}    {'R'}    {'R'}    {'Y'}
    {'G'}    {'B'}    {'G'}    {'Y'}
    {'B'}    {'R'}    {'Y'}    {'B'}
```

Proj4

```
Results =

  4×4 cell array

    {'B'}    {'R'}    {'Y'}    {'Y'}
    {'R'}    {'G'}    {'B'}    {'R'}
    {'B'}    {'Y'}    {'Y'}    {'B'}
    {'G'}    {'G'}    {'G'}    {'Y'}
```

Proj5

```
Results =

  4×4 cell array

    {'Y'}    {'G'}    {'Y'}    {'R'}
    {'Y'}    {'R'}    {'W'}    {'B'}
    {'B'}    {'G'}    {'B'}    {'B'}
    {'G'}    {'B'}    {'G'}    {'R'}
```

Proj1_1

```
Results =

  4×4 cell array

    {'W'}    {'Y'}    {'B'}    {'W'}
    {'W'}    {'W'}    {'W'}    {'W'}
    {'B'}    {'G'}    {'R'}    {'B'}
    {'R'}    {'B'}    {'Y'}    {'R'}
```

Proj1_2

```
Results =

  4×4 cell array

    {'G'}    {'R'}    {'Y'}    {'B'}
    {'Y'}    {'G'}    {'B'}    {'Y'}
    {'R'}    {'G'}    {'W'}    {'Y'}
    {'B'}    {'Y'}    {'W'}    {'W'}
```

Proj1_3

```
Results =

  4×4 cell array

    {'W'}    {'R'}    {'G'}    {'B'}
    {'B'}    {'G'}    {'B'}    {'W'}
    {'B'}    {'B'}    {'W'}    {'B'}
    {'W'}    {'B'}    {'B'}    {'G'}
```

Proj1_4

```
Results =

  4×4 cell array

    {'B'}    {'B'}    {'Y'}    {'B'}
    {'Y'}    {'R'}    {'R'}    {'R'}
    {'W'}    {'R'}    {'G'}    {'B'}
    {'R'}    {'R'}    {'R'}    {'G'}
```

Proj1_5

```
Results =

  4×4 cell array

    {'W'}    {'R'}    {'B'}    {'B'}
    {'R'}    {'R'}    {'G'}    {'Y'}
    {'B'}    {'B'}    {'R'}    {'B'}
    {'G'}    {'W'}    {'R'}    {'G'}
```

Proj2_1

```
Results =

  4×4 cell array

    {'R'}    {'R'}    {'G'}    {'Y'}
    {'W'}    {'B'}    {'Y'}    {'B'}
    {'Y'}    {'Y'}    {'B'}    {'W'}
    {'R'}    {'G'}    {'W'}    {'G'}
```

Proj2_2

```
Results =

  4×4 cell array

    {'Y'}    {'B'}    {'B'}    {'G'}
    {'G'}    {'Y'}    {'W'}    {'B'}
    {'B'}    {'G'}    {'G'}    {'B'}
    {'B'}    {'Y'}    {'Y'}    {'R'}
```

Proj2_3

```
Results =

  4×4 cell array

    {'G'}    {'W'}    {'R'}    {'W'}
    {'B'}    {'G'}    {'B'}    {'G'}
    {'W'}    {'Y'}    {'Y'}    {'Y'}
    {'R'}    {'B'}    {'W'}    {'Y'}
```

Proj2_4

```
Results =

  4×4 cell array

    {'Y'}    {'R'}    {'R'}    {'R'}
    {'R'}    {'Y'}    {'G'}    {'Y'}
    {'Y'}    {'G'}    {'B'}    {'Y'}
    {'R'}    {'W'}    {'B'}    {'Y'}
```

Proj2_5

```
Results =

  4×4 cell array

    {'G'}    {'R'}    {'G'}    {'W'}
    {'B'}    {'B'}    {'G'}    {'W'}
    {'R'}    {'W'}    {'R'}    {'Y'}
    {'R'}    {'Y'}    {'Y'}    {'W'}
```

Rot_1

```
Results =

  4×4 cell array

    {'R'}    {'W'}    {'W'}    {'G'}
    {'Y'}    {'W'}    {'R'}    {'W'}
    {'B'}    {'W'}    {'R'}    {'Y'}
    {'Y'}    {'R'}    {'Y'}    {'Y'}
```

Rot_2

```
Results =

  4×4 cell array

    {'G'}    {'W'}    {'Y'}    {'R'}
    {'W'}    {'R'}    {'B'}    {'W'}
    {'B'}    {'B'}    {'G'}    {'W'}
    {'G'}    {'B'}    {'B'}    {'B'}
```

Rot_3

```
Results =

  4×4 cell array

    {'W'}    {'G'}    {'W'}    {'R'}
    {'W'}    {'G'}    {'R'}    {'R'}
    {'W'}    {'G'}    {'G'}    {'R'}
    {'G'}    {'R'}    {'G'}    {'G'}
```

Rot_4

```
Results =

  4×4 cell array

    {'R'}    {'R'}    {'G'}    {'Y'}
    {'G'}    {'Y'}    {'Y'}    {'W'}
    {'W'}    {'W'}    {'R'}    {'B'}
    {'W'}    {'R'}    {'W'}    {'B'}
```

Rot_5

```
Results =

  4×4 cell array

    {'B'}    {'B'}    {'B'}    {'W'}
    {'G'}    {'Y'}    {'R'}    {'G'}
    {'B'}    {'G'}    {'Y'}    {'Y'}
    {'R'}    {'R'}    {'R'}    {'R'}
```