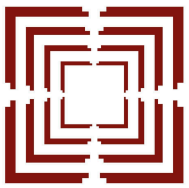


Ahmedabad
University

CromaScan color and pattern recognition

Faculty Advisor: Sanket Patel

Monsoon Semester- 2023

EXECUTIVE SUMMARY:

The project aims to develop a robust image color and pattern defect detection system using a combination of scale-invariant feature transformation (SIFT) with Laplacian optimization and K-means clustering. The project operates within the software and a small physical setup. The software includes a website that captures images, employs SIFT and K-means, and compares them against an existing database, providing accurate image pattern matching for diverse applications. The physical setup includes a C930e Logitech webcam that captures current design images. The algorithm calculates a similarity percentage, facilitating accurate defect identification by quantifying the likeness between the captured and ideal images. With a comprehensive approach to crucial point extraction by SIFT algorithm for pattern recognition and color palette extraction using k-means clustering, the project provides a solution for industries aiming to enhance the precision and efficiency of defect identification in design images.

MOTIVATION BEHIND THE PROJECT:

In quality control and manufacturing, accurately detecting defects in color and pattern plays a pivotal role. With an increasing focus on automatic practices, the need for efficient defect detection aligns with broader initiatives for defect-free production. Inspired by the necessity to enhance quality standards and reduce waste, this project uses advanced image processing techniques, specifically the K-means clustering algorithm and SIFT, to automatically identify colour and pattern defects in manufacturing processes. Pursuing more precise defect detection aligns with a commitment to pointing out imperfections in production, contributing to a responsible manufacturing landscape. In this context, this project seeks to establish a scientific foundation for automatic defect detection in color and patterns, providing valuable insights for future advancements in quality control within the manufacturing sector.

PROJECT INTRODUCTION:

Background

Image processing is a multidisciplinary field that involves the manipulation and analysis of images using various techniques. It plays a crucial role in numerous applications, ranging from medical imaging and satellite imagery to industrial quality control and computer vision systems. The primary goal of image processing is to enhance the quality of images, extract valuable information, and make them suitable for further analysis or interpretation.

In the context of our project, the use of the Scale-Invariant Feature Transformation (SIFT) algorithm and K-means clustering indicates a focus on feature extraction and pattern recognition. These techniques are commonly employed in image processing for tasks such as object recognition, matching, and classification.

By combining these techniques with a webcam-based physical setup, our project aims to

provide a robust solution for image color and pattern defect detection, particularly in design images. This application is valuable in industries where precision and efficiency in defect identification are critical for quality control processes.

Project Objective
Project deliverables
Defect detection of in color and pattern using image processing

Project Description

Methodology

1. **K-Means**

K-means clustering is a data partitioning technique used in image analysis to extract predominant colors and determine image similarity based on color distributions. In this method, images are initially represented as a collection of pixels, each containing color information. To extract colors, the image's pixel data is processed and organized into clusters representing dominant colors.

The process begins with converting the image's color space, often from RGB to HSV or LAB, to better represent colors and separate intensity from color information. The pixel data is then gathered and prepared for clustering. K-means algorithm, an unsupervised learning technique, is applied to this dataset. It groups similar data points into 'k' clusters, with 'k' representing the desired number of colors to be extracted.

Each cluster centroid obtained after the K-means convergence denotes a representative color in the image. These centroids act as the extracted colors. By setting 'k,' the algorithm effectively groups similar colors, quantizing the color space into 'k' dominant colors.

Histograms are constructed for each image to compare images based on these extracted colors. These histograms contain bins representing the frequency or quantity of each color cluster centroid. Metrics like Earth Mover's Distance (EMD) or statistical measures are then employed to quantify the similarity between these histograms. Lower distances indicate higher similarity in color distributions, suggesting resemblances in dominant colors between images. Conversely, more considerable distances imply dissimilarity in color composition.

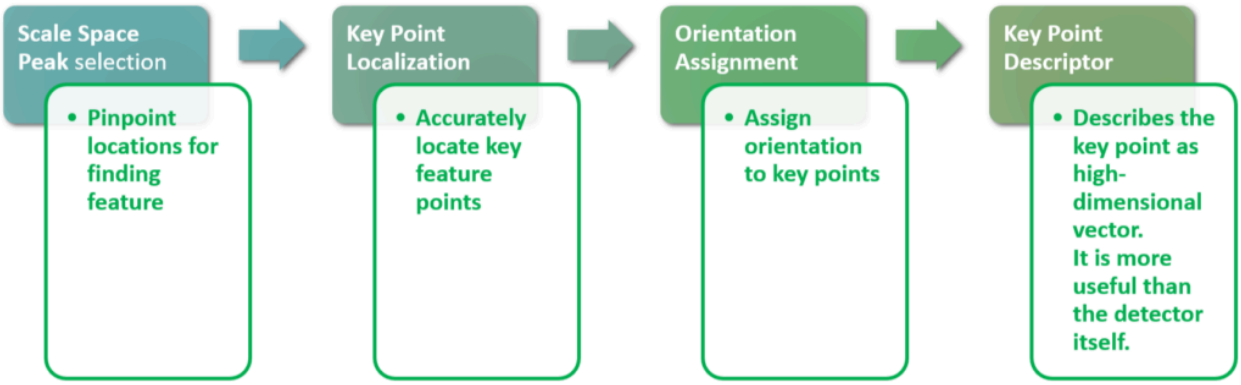
Adjustments in K-means parameters, color space, or similarity metrics can significantly impact the accuracy of image similarity determination. While this method focuses on color distribution likeness between images, it may not encapsulate all aspects of visual similarity, such as texture or structural content. Hence, it is a foundational approach to measure similarity predominantly based on color representations extracted via K-means clustering.

2. **SIFT**

SIFT (Scale Invariant Feature Transform) Detector is used in the detection of interest points on

an input image. It allows the identification of localized features in images which is essential in applications such as:

Object Recognition in Images
Path detection and obstacle avoidance algorithms
Gesture recognition, Mosaic generation, etc.
Unlike the Harris Detector, which is dependent on properties of the image such as viewpoint, depth, and scale, SIFT can perform feature detection independent of these properties of the image. This is achieved by the transformation of the image data into scale-invariant coordinates. The SIFT Detector has been said to be a close approximation of the system used in the primate visual system.



Let's examine the three vital phases in the identification of keypoints and the creation of descriptors using SIFT:

Scale Space Peak Selection and Keypoint Localization:
Scale-space peak selection is a technique the SIFT method uses to locate key points in an image. Several successively blurrier images of various sizes are first produced to do this. The resulting difference-of-Gaussians (DoG) images, produced by deducting neighboring Gaussian-blurred images, display peaks representing areas with discernible scale-to-scale intensity fluctuations. These peaks could be important points of reference.

Using the scale space, a series of progressively blurred images, SIFT can identify key points that are consistently noticeable at different object sizes and camera distances. The DoG pictures' keypoints are identified as local maxima or minima, highlighting distinct features at various scales. The initial DoG extrema is refined with a Taylor series approximation to guarantee accurate localization. This meticulous process enhances the accuracy of pinpointing key points in the image.

Assigning Orientation for Robustness:
Certain features, such as corners or edges, should retain relative orientation when an item is rotated within an image. To solve this, SIFT examines the gradients surrounding each key point, showing the direction and size of intensity variations. Through the computation of gradients at various scales in the vicinity of the keypoint, SIFT creates a histogram that represents the prevailing gradient direction, thereby ascertaining the orientation of the keypoint. This

orientation invariance ensures a solid comparison to picture rotations since keypoints with comparable local appearances also have similar exposures independent of the total image rotation.

Encoding Local Appearance through Descriptors:
SIFT generates a 128-dimensional descriptor to capture the distinct local appearance around each keypoint and facilitate matching under different lighting conditions or small deformations. This descriptor partitions the keypoint neighborhood into smaller sections to encode local image information. SIFT computes a histogram of gradient orientations inside each sub-region to represent the regional distribution of gradient directions. The 128-dimensional descriptor is then formed by concatenating the histograms from each sub-region, which produces a unique depiction of the regional image appearance around the keypoint. Interestingly, gradient information is more resilient to changes in light than pixel intensity values, improving SIFT descriptors' resilience.

SIFT is exceptionally effective at matching keypoints between pictures with high reliability by integrating orientation assignment, keypoint localization, and descriptor creation. This capacity is quite helpful for object recognition as it allows precise detection by recognising and matching important traits between photos of the object and the scene. Furthermore, SIFT makes it easier for key points in numerous photos to fit seamlessly during image stitching, which helps to create panoramic views.

Expected Benefits

Efficient Defect Detection:
The project uses two algorithms together to quickly spot color and pattern differences, making sure defects are found correctly. SIFT finds features while K-means groups similar things. This automatic way cuts down mistakes people could make, giving a smooth and right way to see problems as they happen in real factory work.

Automated Comparison:
Using K-means and SIFT algorithms, the system can compare captured photos against a pre-existing database. Automatically comparing these documents speeds up the process of inspections, especially important where there are many products per unit of time and less human labor and time are required while keeping precision.

Enhanced Accuracy:
SIFT and K-means algorithms improve defect detection accuracy through deep color and pattern feature analysis. This integrated approach is superior to traditional methods and provides reliable results, ensuring heart defect detection in different visual contexts.

Removal of Environmental Noise:
To provide a sharp, targeted examination of image data, the system has the faculty to disregard contextual randomness. The initiative heightens the correctness of flaw recognition, decreases unfounded judgments, and strengthens the trustworthiness of the review procedure by removing

non essential factors.

Improved Quality Control:
By detailed observation, the system aids in sustaining superb manufacturing standards by discerning issues, decreasing below-average output, reducing waste, and making sure of consistent high-quality. This eventually results in an elevated degree of quality results.

Cost Efficiency:
We can prevent producing defective things by identifying errors early on via automation, decreasing the costs of following repairs or recreating products. This cutting costs stage clarifies manufacturing procedures and improves the efficiency of a business.

Project Scope
Main aim of this project is to detect anomalies in colors and pattern by comparing a sample to the normal using K-means clustering and.SIFT algorithm

LIST OF COMPONENTS:

webcam

Name	**Specifications**
Webcam	C930E BUSINESS WEBCAM

Feature	Description
Video Resolution	1080p (Full HD) @ 30fps
Field of View (dFOV)	90° diagonal
Digital Zoom	4x digital zoom (Full HD) available
Autofocus	Built-in HD autofocus

Light Correction	RightLight 2 auto light correction for clear image in various lighting environments
Video Compression	H.264 UVC 1.5 with Scalable Video Coding for a smoother video stream in applications
Audio	Dual omni-directional mics, optimized to capture audio clearly from up to one meter away
Connectivity	Easily connects via USB-A; cable length of 5 ft (1.5 m)
Privacy Shutter	Switch to privacy mode with the attachable privacy shade that flips up and down
Mounting Options	Universal clip; 1/4” thread for tripod mounting (Tripod not included)
Software Support	Logi Tune available at www.logitech.com/Tune to control zoom, adjust color, set manual focus, and update firmware
Certifications	Certified for Microsoft Teams® and Skype™ for Business
Compatibility	Compatible with Microsoft DirectShow. Works with other common calling applications such as BlueJeans, Cisco Webex™, Fuze, Google Meet™, GoToMeeting®, Zoom®, and others
Dimensions & Weight	Height x Width x Depth: 1.69 in (43mm) x 3.7 in (94 mm) x 2.80 in (71 mm) Weight: 5.71 oz (162 g)
Cable Length	5 ft (1.5 m) USB-A cable
Warranty	3 years

EXPLANATION OF THE CIRCUIT/WORKING/FLOW CHARTS:

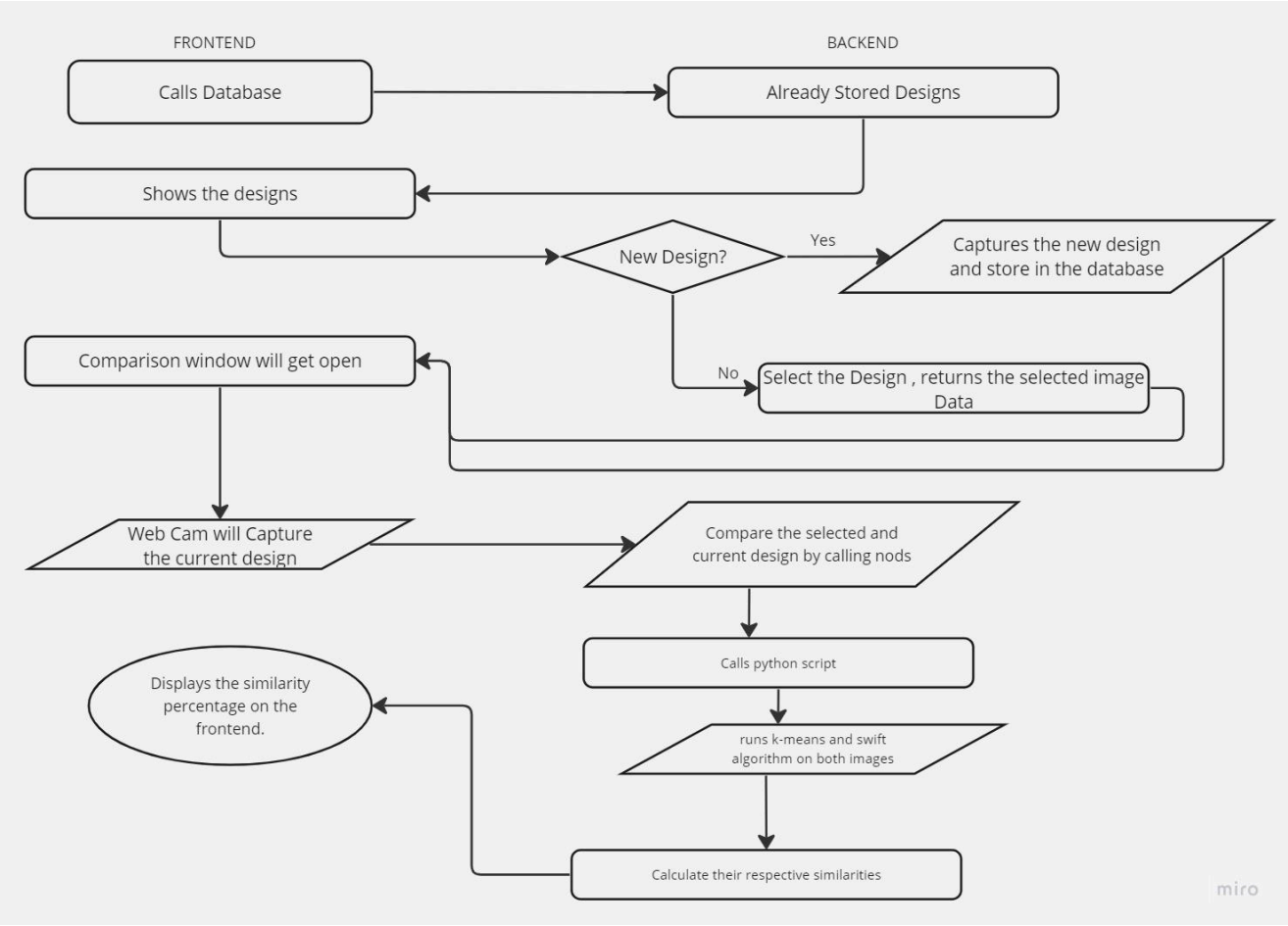


Figure 1. Flow chart of backend and frontend communication

The flowchart shows the communication between frontend and backend. First the backend calls the database for the already existing design we can select one design to which we want to compare. Then the next screen appears with a live camera feed coming from the webcam and by clicking on the compare the backend is called to run the python scripts through API for color and pattern detection, we can compare current design with the design we choose from the database. On the main screen there is also a option to add new design by which we can add a new design to the database.

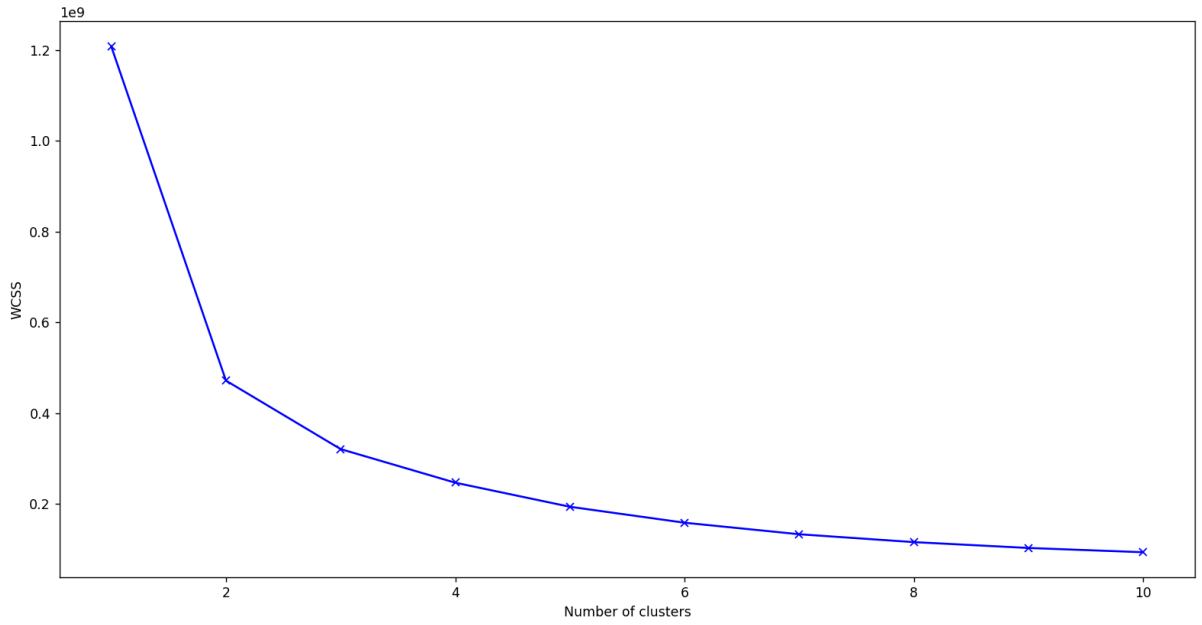
MEASUREMENT AND CALIBRATION:

Elbow Method Optimization in K-Means Clustering:
The Elbow Method determines the optimal number of clusters (k) in a K-means clustering algorithm. K-means is an unsupervised machine learning algorithm that partitions a dataset into k clusters based on similarity. The Elbow Method is employed to find the point at which the

within-cluster sum of squares (WCSS) starts to show diminishing returns, resembling an "elbow" in the plot of WCSS against the number of clusters.

Steps in Elbow Method Optimization:

1. **Vary the Number of Clusters (k):** Begin by running the K-means algorithm on the dataset for a range of values of k. It involves partitioning the data into different numbers of clusters, from k=1 to a specific maximum value.
2. **Calculate WCSS for Each k:** For each value of k, calculate the WCSS, which is the sum of the squared distances between each point in a cluster and its centroid. WCSS measures how compact the groups are – lower values indicate tighter, more well-defined clusters.
3. **Plot the Elbow Curve:** Plot the calculated WCSS values against the corresponding values of k. The plot often resembles an arm, and the "elbow" of the curve is the point where adding more clusters ceases to reduce the WCSS significantly.



4. **Identify the Elbow Point:** The Elbow point represents the optimal number of clusters. It is the point where the addition of more collections provides diminishing returns in terms of reducing WCSS. The Elbow point signifies a trade-off between the complexity of having more clusters and the gain in intra-cluster cohesion.
5. **Select the Optimal k:** Choose the k value corresponding to the Elbow point as the optimal number of clusters for the dataset.

Interpretation and Considerations:

- **Elbow Point Interpretation:** The Elbow point is where the rate of decrease in WCSS sharply changes, forming an angle resembling an elbow. It indicates the point of diminishing returns, suggesting that adding more clusters does not significantly improve the model's ability to explain variance in the data.
- **Balance Between Complexity and Performance:** The Elbow Method provides a practical balance between the complexity of the clustering model (more clusters) and its

performance (reduction in WCSS). It helps prevent overfitting by avoiding an excessively detailed partitioning of the data.

- **Reassessing with Domain Knowledge:** While the Elbow Method is a valuable heuristic, it is advisable to reassess the choice of k with domain knowledge and, if available, other validation metrics for clustering performance.

Down Sampling

Downsampling refers to the process of reducing the size or resolution of an image. It involves removing some pixel data from the picture, resulting in a lower resolution or smaller version of the original image. Downsampling is commonly used for various purposes, including reducing storage requirements, speeding up processing, or preparing images for display on different devices.

Downsampling is used here to create more miniature representations of the original images. This downscaled representation might be utilized for various purposes, such as reducing computational complexity in subsequent image processing or analysis steps.
Downsampling is employed before calculating the similarity between images based on color histograms. This process is used as a preprocessing step to reduce the complexity of the color information while maintaining a level of representative data for comparison purposes.

REAL-WORLD APPLICATION OF THE PROJECT:

Pattern Detection:

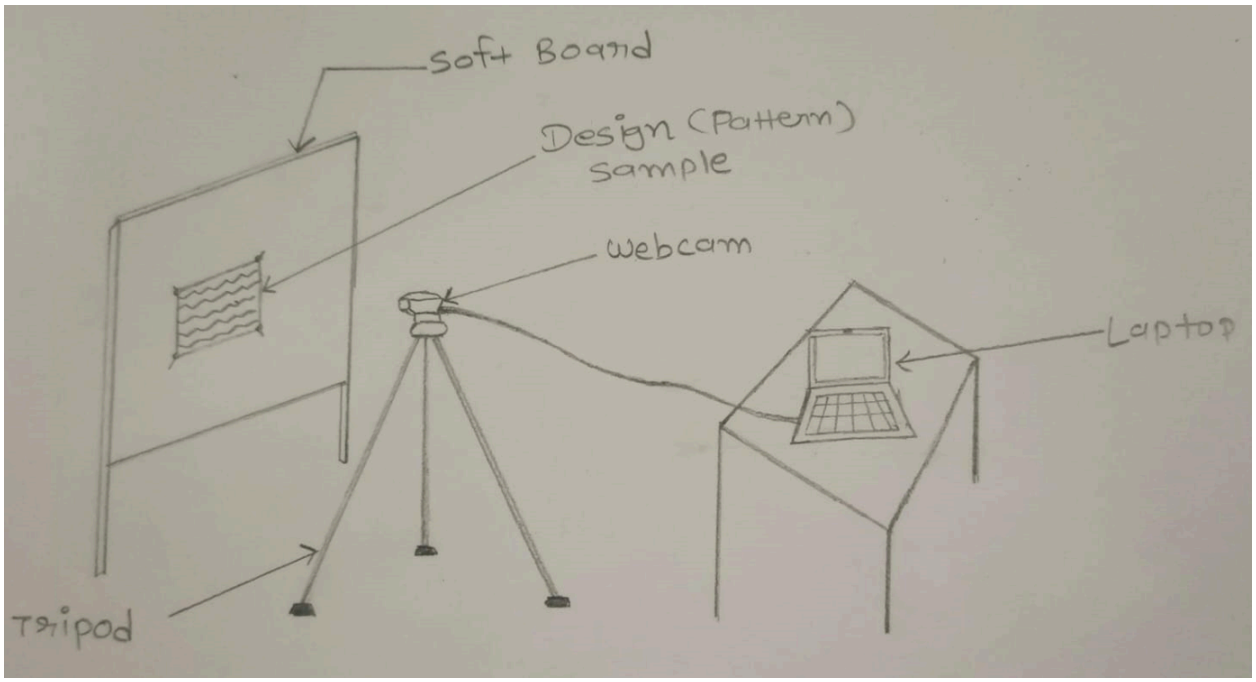
Textiles and Printing: Ensures superior textiles and printed materials by spotting defects or irregularities in fabric patterns.
Electronics Manufacturing: Verifies accuracy and functionality by identifying flaws in printed circuit boards (PCBs) or componentry.
Medical imaging: Assists in diagnosis and treatment planning by detecting anomalies or irregularities in medical images.
Art restoration: Supports restoration efforts by helping to spot defects or other irregularities in artwork.
Plywood Industry: Detect defects in the plywood and automate the entire chain

Colour Detection:

Quality Control in Manufacturing: Maintaining brand standards through quality control in manufacturing, products such as packaging, labels, or printed materials are guaranteed to have consistent color.
Pharmaceuticals: Assures accurate dosages and prevents errors by checking color consistency in pills and capsules.
Food Industry: Verifies food product color consistency, labeling, and packaging for consumer safety and trust.

Automotive and Aerospace: Consistent coloration in parts is ensured in the automotive and aerospace industries, improving their aesthetics and quality.

DRAWINGS OF THE EXPERIMENTAL SETUPS:



TOTAL COST TO DEVELOP THE PROJECT:

ITEMS	**COST**
Camera	10700
Total	**10700**

SUMMARY,CONCLUSION AND FUTURE WORK:

Summary:

We use KNN and K means clustering to detect the color and patterns in an object (preferably a sheet of ply). We also use The Elbow Method to improve the speed and accuracy of the detection. We have connected this to a website and a setup. The setup makes a favorable environment with all the necessary favorable conditions, so that there is very less room for error. By connecting it to the website, we can store some reference patterns and compare the mass with them. The results are also displayed on the website.

Conclusion:
We have successfully made an application that detects the defects in pattern and colors with % accuracy using K-means and SIFT algorithm. We also have connected everything to the best with the resources provided.

Future Work:
The website is currently working on a local system only and the online deployed version is very slow so we could make it as fast and responsive as it works on the local system.

CODES:

K-Means for Extracting color and comparing:

Imports

```
import cv2
import numpy as np
from sklearn.cluster import KMeans
from scipy.stats import wasserstein_distance
import base64
import json
import sys
```

Image Processing Functions

"downsample_image(image, factor)"
Resizes an image using OpenCV's resize function, taking in a picture and a downscaling factor.

```
def downsample_image(image, factor):
    return cv2.resize(image, (0, 0), fx=factor, fy=factor)
```

"extract_colors_from_image(image, num_colors)"
Converts the image to the HSV color space.

- Applies KMeans clustering to find dominant colors.
- Calculates color percentages and sorts them based on occurrence.

```
def extract_colors_from_image(image, num_colors):
    pixels = cv2.cvtColor(image, cv2.COLOR_BGR2HSV).reshape(-1, 3)

    kmeans = KMeans(n_clusters=num_colors).fit(pixels)

    counts = np.bincount(kmeans.labels_)
    percentages = (counts / counts.sum()) * 100

    color_data = sorted(zip(percentages, kmeans.cluster_centers_),
key=lambda x: x[0], reverse=True)
```

```
    return color_data
```

“calculate_color_similarity(image1, image2, num_colors)”

- Extract dominant colors from two images using extract_colors_from_image.
- Computes the Earth Mover's Distance (EMD) between histograms of dominant colors from the pictures.

```
def calculate_color_similarity(image1, image2, num_colors):
    colors1 = extract_colors_from_image(image1, num_colors)
    colors2 = extract_colors_from_image(image2, num_colors)

    hist1 = [color[0] for color in colors1]
    hist2 = [color[0] for color in colors2]

    emd_distance = wasserstein_distance(hist1, hist2)

    return emd_distance
```

“calculate_similarity_percentage(emd_distance, max_distance)”
Calculates the similarity percentage based on the EMD score and a maximum distance threshold.

```
def calculate_similarity_percentage(emd_distance, max_distance):
    similarity_percentage = 100 * (1 - (emd_distance / max_distance))
    return similarity_percentage
```

Main Execution

1. Reading and Decoding Images
    - Reads base64-encoded images from JSON input and attempts to decode them.
    - Catches exceptions during image decoding.

```
input_data = json.load(sys.stdin)
    image_base64_1 = input_data["image_base64_1"]
    image_base64_2 = input_data["image_base64_2"]
    num_colors = 5
    error = None
    downsample_factor = 0.5

    try:
        image1 =
cv2.imdecode(np.frombuffer(base64.b64decode(image_base64_1), np.uint8),
cv2.IMREAD_COLOR)
```

```
        image2 =
cv2.imdecode(np.frombuffer(base64.b64decode(image_base64_2), np.uint8),
cv2.IMREAD_COLOR)
    except Exception as e:
        print(f"Error decoding images: {e}")
        image1, image2 = None, None
```

2. Image Processing
    - If images are successfully decoded:
        - Downsampled the images.
        - Sets the maximum possible distance for similarity.

```
    if image1 is not None and image2 is not None:
        # Downsample the images
        downsampled_image1 = downsample_image(image1, downsample_factor)
        downsampled_image2 = downsample_image(image2, downsample_factor)

        max_possible_distance = 100.0  # Adjust this based on your
requirement
```

3. Calculating Similarity
    - Calls calculate_color_similarity to get the similarity score.
    - Handles different scenarios:
        - Calculate and round the similarity percentage if the score is under a certain threshold (here, <2).
        - If the similarity score is above the point, it determines that the images are not similar.
        - If errors occur during the similarity calculation or image decoding, it sets an error message.

```
similarity_score = calculate_color_similarity(downsampled_image1,
downsampled_image2, num_colors)
        if similarity_score is not None:
            if similarity_score < 2:
                similarity_percentage =
calculate_similarity_percentage(similarity_score, max_possible_distance)
                similarity_percentage = round(similarity_percentage,2)
                if similarity_percentage is not None:
                    similarity_percentage = f"{similarity_percentage}%"
                else:
                    error ="Error calculating similarity percentage."
            else:
                similarity_percentage = "Image not similar"
        else:
```

```
            error = "Error calculating similarity score."
    else:
        error = "Images not properly decoded."
```

4. Output as JSON
    - Creates a JSON object containing the similarity score, percentage, and any potential errors encountered.
    - Prints the JSON object to the standard work.

```
result = {
        'similarity_score': round(similarity_score,2),
        'similarity_percentage': similarity_percentage,
        'error': error
    }
    json.dumps(result)
    print(json.dumps(result))
```

Summary
This script receives two base64-encoded images, processes them to extract dominant colors, computes their color similarity using the Earth Mover's Distance (EMD), and generates a JSON response containing the similarity score, percentage, and any potential errors encountered during the process.

SIFT CODE:

Imports

```
import cv2
import numpy as np
import base64
import json
import sys
```

Image Decoding from Base64

```
def decode_base64_image(base64_string):
    base64_bytes = base64.b64decode(base64_string)
    image = np.frombuffer(base64_bytes, dtype=np.uint8)
    image = cv2.imdecode(image, cv2.IMREAD_COLOR)
    return image
```

- base64.b64decode(base64_string): Decodes the base64 string into bytes.

- np.frombuffer(base64_bytes, dtype=np.uint8): Converts the bytes into a NumPy array of type uint8.
- cv2.imdecode(image, cv2.IMREAD_COLOR): Reads the image data into an OpenCV format using the IMREAD_COLOR flag.

Image Similarity Calculation

```
def calculate_image_similarity(base64_string1, base64_string2):
    image1 = decode_base64_image(base64_string1)
    image2 = decode_base64_image(base64_string2)



    max_height = 1080
    scaling_factor = max_height / max(image1.shape[0], image2.shape[0])
    image1 = cv2.resize(image1, (0, 0), fx=scaling_factor,
fy=scaling_factor)
    image2 = cv2.resize(image2, (0, 0), fx=scaling_factor,
fy=scaling_factor)

    sift = cv2.SIFT_create()

    keypoints1, descriptors1 = sift.detectAndCompute(cv2.cvtColor(image1,
cv2.COLOR_BGR2GRAY), None)
    keypoints2, descriptors2 = sift.detectAndCompute(cv2.cvtColor(image2,
cv2.COLOR_BGR2GRAY), None)

    bf = cv2.BFMatcher()

    matches = bf.knnMatch(descriptors1, descriptors2, k=2)

    good_matches = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            good_matches.append(m)

    if len(good_matches) > 4:
        src_pts = np.float32([keypoints1[m.queryIdx].pt for m in
good_matches]).reshape(-1, 1, 2)
        dst_pts = np.float32([keypoints2[m.trainIdx].pt for m in
good_matches]).reshape(-1, 1, 2)
```

```
        homography, mask = cv2.findHomography(src_pts, dst_pts,
cv2.RANSAC, 5.0)

        inlier_percentage = (np.sum(mask) / len(good_matches)) * 100
        inlier_percentage = round(inlier_percentage,2)
        similarity_threshold = 30.0

        if inlier_percentage >= similarity_threshold:
            return {"result": "Images are similar",
"similarity_percentage": f"{inlier_percentage} %"}
        else:
            return {"result": "Images are not similar",
"similarity_percentage": f"{inlier_percentage} %"}
    else:
        return {"result": "Not enough matches to determine similarity",
"similarity_percentage": "Image not similar"}
```

- Image Resizing: Ensures both images have a consistent height for comparison.
- SIFT Feature Detection and Description: Detects key points and computes descriptors for both images using the SIFT algorithm.
- Feature Matching and Filtering: Matches key points between images and filters based on a distance ratio test to get good matches.
- Homography and Similarity Calculation: Calculates a homography matrix, then computes the percentage of inliers. Determines image similarity based on the inlier percentage and a similarity threshold.

Main Execution

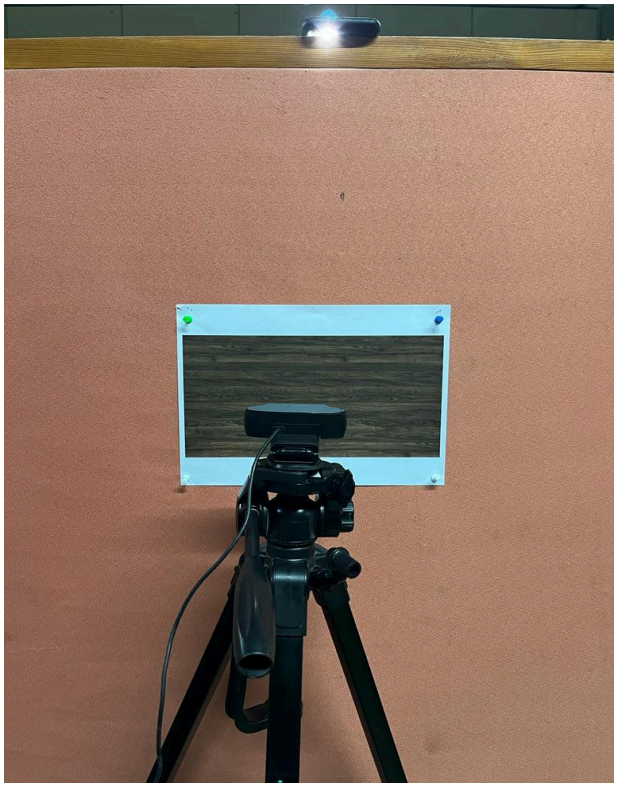
```
if __name__ == "__main__":
    input_data = json.load(sys.stdin)
    base64_string1 = input_data["base64_string1"]
    base64_string2 = input_data["base64_string2"]

    result = calculate_image_similarity(base64_string1, base64_string2)

    print(json.dumps(result))
```

- JSON Input: Reads JSON data from the standard input containing base64 encoded strings for two images.
- Image Comparison: Calls the calculate_image_similarity function with the provided base64 strings.
- JSON Output: Prints the comparison results as a JSON object to the standard output

PHOTOGRAPHS(CIRCUITS/MEASUREMENTS/etc.):



References:

https://www.logitech.com/content/dam/logitech/en_us/video-collaboration/pdf/c930e-datasheet.pdf

https://www.geeksforgeeks.org/sift-interest-point-detector-using-python-opencv/

https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/