Republic of the Philippines
Laguna State Polytechnic University
Province of Laguna

Machine Problem No. 3			
Topic:	**Module 2.0: Feature Extraction and Object Detection**	**Week No.**	6-7
Course Code:	**CSST106**	**Term:**	1st Semester
Course Title:	**Perception and Computer Vision**	**Academic Year:**	2024-2025
Student Name	Maxyne Nuela Ignacio	**Section**	BSCS-IS-4B
Due date		**Points**	

Machine Problem No. 3: Feature Extraction and Object Detection

Objective:
The objective of this machine problem is to implement and compare the three feature extraction methods (**SIFT**, **SURF**, and **ORB**) in a single task. You will use these methods for feature matching between two images, then perform image alignment using **Homography** to warp one image onto the other.

Problem Description:
You are tasked with loading two images and performing the following steps:

1. Extract keypoints and descriptors from both images using **SIFT**, **SURF**, and **ORB**.
2. Perform feature matching between the two images using both **Brute-Force Matcher** and **FLANN Matcher**.
3. Use the matched keypoints to calculate a **Homography matrix** and align the two images.
4. Compare the performance of SIFT, SURF, and ORB in terms of feature matching accuracy and speed.

You will submit your code, processed images, and a short report comparing the results.

Task Breakdown:

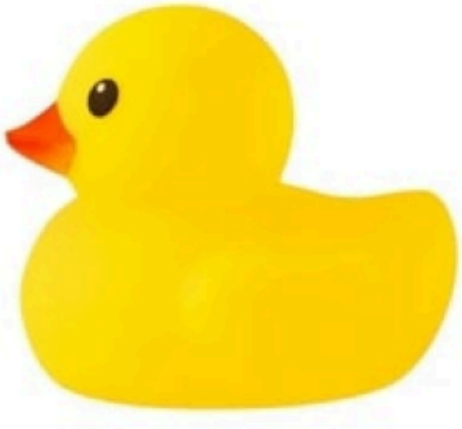
Step 1: Load Images

- Load two images of your choice that depict the same scene or object but from different angles.

```python
from google.colab import drive
import cv2
import matplotlib.pyplot as plt

# Mount Google Drive
drive.mount('/content/drive')

# Set paths to your images in Google Drive
image1_path = '/content/drive/My Drive/01-MP3.jpg'
image2_path = '/content/drive/My Drive/02-MP3.jpg'

# Load the images using OpenCV
image1 = cv2.imread(image1_path)
image2 = cv2.imread(image2_path)

# Check if images are loaded correctly
if image1 is None or image2 is None:
    print("Error loading images.")
    if image1 is None:
        print("Image 1 could not be loaded. Check the file path.")
    if image2 is None:
        print("Image 2 could not be loaded. Check the file path.")
else:
    # Convert BGR images to RGB for displaying with Matplotlib
    image1_rgb = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
    image2_rgb = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)

    # Display the images side-by-side
    plt.figure(figsize=(10, 5))
```

Image 1

Image 2



➔ This code mounts Google Drive, loads two images from specified paths, and checks if they load successfully. If they are loaded, it converts them from BGR to RGB (for accurate colors with

Matplotlib) and displays them side-by-side. If either image fails to load, it notifies the user about which image could not be loaded.

Step 2: Extract descriptors.

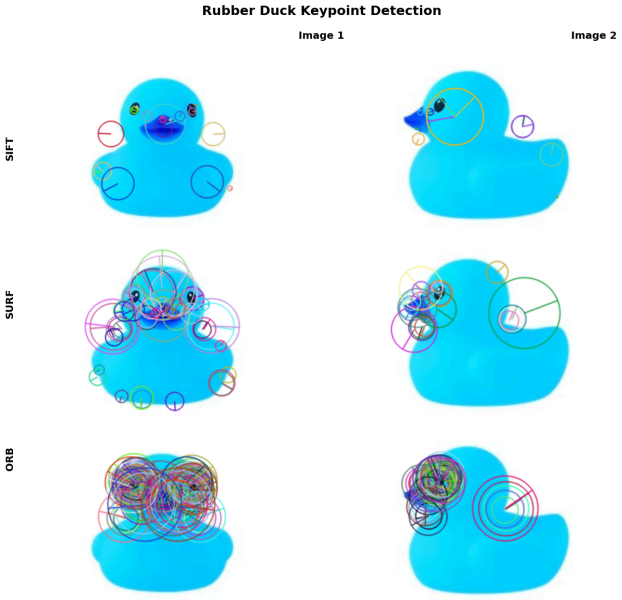
Keypoints and Descriptors Using SIFT, SURF, and ORB

- Apply the **SIFT** algorithm to detect keypoints and compute descriptors for both images.
- Apply the **SURF** algorithm to do the same.
- Finally, apply **ORB** to extract keypoints and

Submission:

- Python code (feature_extraction.py)
- Processed images showing keypoints for SIFT, SURF, and ORB (e.g., sift_keypoints.jpg, surf_keypoints.jpg, orb_keypoints.jpg).

```
#Function to draw keypoints.
def draw_keypoints(image, keypoints):
    return cv2.drawKeypoints(image, keypoints, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

#Draw keypoints for SIFT.
image1_sift_kp = draw_keypoints(image1, sift_features[0])
image2_sift_kp = draw_keypoints(image2, sift_features[2])

#Draw keypoints for SURF.
image1_surf_kp = draw_keypoints(image1, surf_features[0])
image2_surf_kp = draw_keypoints(image2, surf_features[2])

#Draw keypoints for ORB.
image1_orb_kp = draw_keypoints(image1, orb_features[0])
image2_orb_kp = draw_keypoints(image2, orb_features[2])

# Display the keypoints for each method
fig, axs = plt.subplots(3, 2, figsize=(15, 12))

# Set a main title for the entire figure
fig.suptitle("Rubber Duck Keypoint Detection", fontsize=18, fontweight='bold')

# Row titles
fig.text(0.1, 0.75, "SIFT", ha='center', va='center', rotation='vertical', fontsize=14, fontweight='bold')
fig.text(0.1, 0.5, "SURF", ha='center', va='center', rotation='vertical', fontsize=14, fontweight='bold')
fig.text(0.1, 0.25, "ORB", ha='center', va='center', rotation='vertical', fontsize=14, fontweight='bold')
```



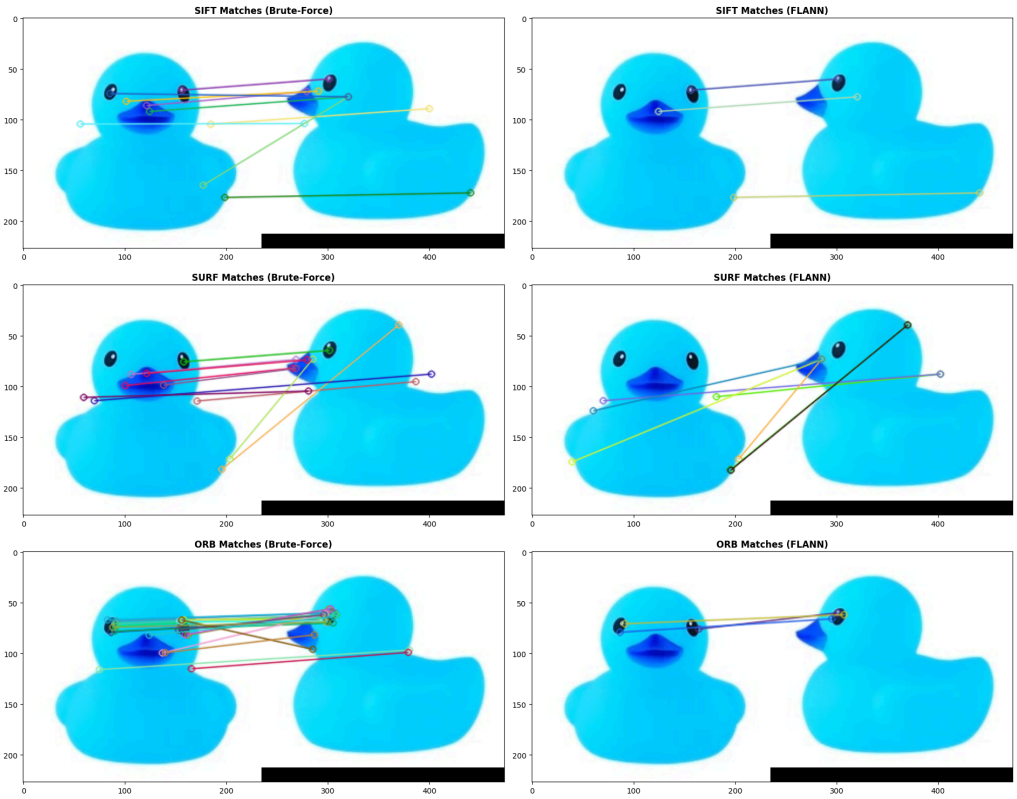
➔ This code displays keypoints detected by three feature-detection methods (SIFT, SURF, and ORB) on two images. It creates a subplot grid with labeled rows for each method and labeled columns for each image, displaying the images side-by-side with keypoints drawn. Finally, it adjusts the layout for a main title, row, and column labels, then shows the plot.

Step 3: Feature Matching with Brute-Force and FLANN

- Match the descriptors between the two images using **Brute-Force Matcher**.
- Repeat the process using the **FLANN Matcher**.
- For each matching method, display the matches with lines connecting corresponding keypoints between the two images.

Submission:

- Python code (feature_matching.py)
- Processed images showing matches for Brute-Force and FLANN for each algorithm (e.g., sift_bf_match.jpg, sift_flann_match.jpg).



➔ This image shows comparisons of feature-matching results between two images of rubber ducks using different keypoint detection methods and matching algorithms. The rows represent different methods (SIFT, SURF, and ORB), and each pair of images in a row displays the matched keypoints between the two rubber duck images. The left side of each row uses a brute-force matcher, while the right side uses the FLANN (Fast Library for Approximate Nearest Neighbors)

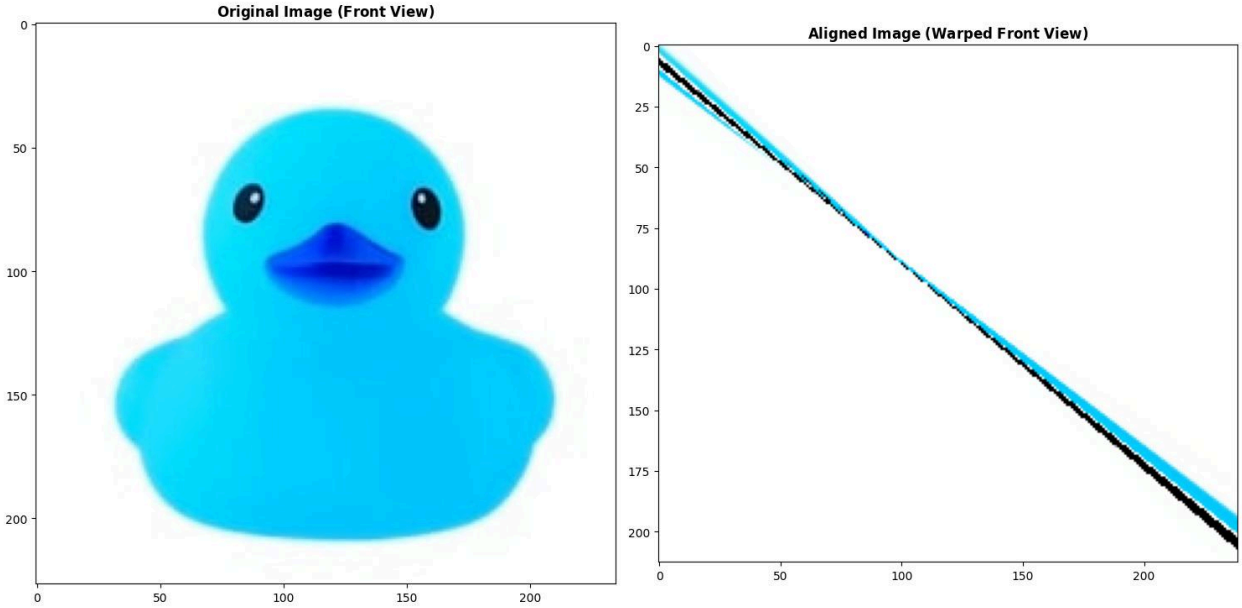
matcher, indicated by the titles above each pair. The colored lines link matched keypoints between the two images, demonstrating each method's accuracy and robustness in finding similar features across the two images.

Step 4: Image Alignment Using Homography

- Use the matched keypoints from **SIFT** (or any other method) to compute a **homography matrix**.
- Use this matrix to warp one image onto the other.
- Display and save the aligned and warped images.

Submission:

- Python code (image_alignment.py)
- Aligned and warped images (e.g., aligned_image.jpg, warped_image.jpg).



➔ The code used in this part aligns one image with another using SIFT keypoints, the Brute-Force (BF) matching technique, and homography transformation. First, it converts each image to grayscale and extracts SIFT keypoints and descriptors, identifying distinctive features in each image. Using a BFMatcher with L2 norm, it matches descriptors between the two images, sorting and retaining the best matches based on distance. From these matched points, it computes a homography matrix using RANSAC, which allows it to warp the first image to align with the second image. The result is displayed side-by-side, showing the original and the aligned (warped) version for comparison.

➔

Step 5: Performance Analysis

1. **Compare the Results**:

- Analyze the performance of **SIFT**, **SURF**, and **ORB** in terms of keypoint detection accuracy, number of keypoints detected, and speed.
- Comment on the effectiveness of **Brute-Force Matcher** versus **FLANN Matcher** for feature matching.

```
Feature Detection Performance Analysis:
Method  Keypoints Detected (Image 1)  Keypoints Detected (Image 2)  Time Taken (s)
  SIFT                           25                            14        0.033574
  SURF                           46                            23        0.033629
   ORB                          119                            54        0.004946

Feature Matching Performance Analysis:
Method  Matches (Brute-Force)  Time Taken (BF)  Matches (FLANN)  Time Taken (FLANN)
  SIFT                      9         0.001587                3            0.000604
  SURF                     10         0.001479                7            0.001054
   ORB                     19         0.001698                3            0.003907
```

➔ This analysis compares the performance of SIFT, SURF, and ORB methods in feature detection and matching across two images. In feature detection, SIFT detected 25 keypoints in the first image and 14 in the second, taking 0.0336 seconds. SURF detected more keypoints, with 46 in Image 1 and 23 in Image 2, in a similar time of 0.0336 seconds. ORB, however, detected significantly more keypoints—119 in Image 1 and 54 in Image 2 in only 0.0049 seconds, indicating its efficiency in quickly identifying numerous features. In feature matching, the Brute-Force (BF) matcher yielded 9 matches for SIFT in 0.0016 seconds, 10 for SURF in 0.0015 seconds, and 19 for ORB in 0.0017 seconds. With the FLANN matcher, SIFT found only 3 matches in 0.0006 seconds, SURF achieved 7 matches in 0.0011 seconds, and ORB found 3 matches in 0.0039 seconds. Overall, ORB was the fastest in feature detection and produced the most keypoints, while SIFT and SURF showed better performance in matching accuracy with fewer keypoints.

2. **Write a Short Report**:
    - Include your observations and conclusions on the best feature extraction and matching technique for the given images.

➔ ORB detected the most keypoints (119 in Image 1 and 54 in Image 2) and was the fastest (0.0049 seconds).

➔ SURF detected a moderate number of keypoints (46 in Image 1 and 23 in Image 2) with a time of 0.0336 seconds.

➔ SIFT detected the fewest keypoints (25 in Image 1 and 14 in Image 2) and took 0.0336 seconds.

➔ With Brute-Force (BF), ORB had the most matches (19), followed by SURF (10) and SIFT (9).

➔ With FLANN, SURF had the most matches (7), while ORB and SIFT had only 3 matches each.

➔ BF matching was faster than FLANN for all methods.

➔ Conclusion: ORB is the fastest and detects the most keypoints, ideal for speed-focused tasks; SURF balances performance and accuracy, making it the best choice for accuracy-focused tasks.

Submission:

- A PDF or markdown document (performance_analysis.pdf or performance_analysis.md).

Submission Guidelines:

- **GitHub Repository**:
  - Create a folder in your CSST106-Perception and Computer Vision repository named Feature-Extraction-Machine-Problem.
  - Upload all code, images, and reports to this folder.

- **File Naming Format**: **[SECTION-LASTNAME-MP3]** 4D-LASTNAME-MP3
  - 4D-BERNARDINO-SIFT.py
  - 4D-BERNARDINO-Matching.jpg

Additional Penalties:

- **Incorrect Filename Format**: -5 points
- **Late Submission**: -5 points per day
- **Cheating/Plagiarism**: Zero points for the entire task

Rubric for Feature Extraction and Object Detection Machine Problem

Criteria	Excellent (90-100%)	Good (75-89%)	Satisfactory (60-74%)	Needs Improvement (0-59%)
Step 2: Feature Extraction (SIFT, SURF, ORB)	All feature extraction methods (SIFT, SURF, ORB) are implemented correctly. The extracted keypoints are clearly visualized and well explained. The code is well-commented, and outputs are saved properly.	Feature extraction is implemented correctly, but there may be minor visualization issues or explanations lacking depth.	At least two methods are implemented correctly, with basic explanations and some issues with visualization or code.	Feature extraction methods are incomplete, implemented incorrectly, or not explained well. Poor or no visualization provided.
Step 3: Feature Matching (Brute-Force and FLANN)	Both Brute-Force and FLANN matchers are implemented correctly, and keypoint matches are clearly visualized with detailed explanations. The matching performance for each method is analyzed.	Both matchers are implemented correctly, but there may be minor issues with the visualization, or the explanation lacks depth.	At least one matcher is implemented correctly, with basic explanations and minimal analysis of matching performance.	Feature matching methods are incomplete, implemented incorrectly, or poorly explained. Matches are not visualized, or results are unclear.
Step 4: Image Alignment Using Homography	The Homography matrix is computed correctly using matched keypoints, and the image is aligned and warped successfully. The output is visually accurate, and the process is well explained.	The Homography matrix is computed correctly, but the alignment has minor issues, or the explanation lacks depth.	The Homography matrix is computed, but there are significant alignment issues, or the explanation is basic.	Homography computation is incorrect or incomplete. Image alignment does not work as expected, or no explanation is provided.
Step 5: Performance Analysis	The performance analysis is thorough, comparing the accuracy and speed of SIFT, SURF, and ORB, and evaluating the effectiveness of Brute-Force and FLANN. The conclusion is insightful and well-supported.	The performance analysis is good but lacks some depth in comparing the methods or has minor gaps in the evaluation of the matchers.	The performance analysis is basic, with minimal comparison or weak conclusions. Some methods or matchers are not evaluated.	The performance analysis is incomplete or missing. Little to no comparison or evaluation of methods and matchers is provided.