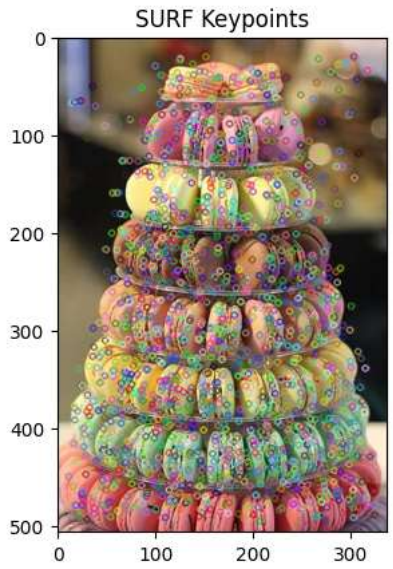
*FOR INSTALLATION OF OPENCV_CONTRIB*

```
!apt-get update
!apt-get install -y cmake build-essential pkg-config

!git clone https://github.com/opencv/opencv.git
!git clone https://github.com/opencv/opencv_contrib.git

!mkdir -p opencv/build
%cd opencv/build
!cmake -D CMAKE_BUILD_TYPE=RELEASE \
        -D CMAKE_INSTALL_PREFIX=/usr/local \
        -D OPENCV_ENABLE_NONFREE=ON \
        -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules \
        -D BUILD_EXAMPLES=OFF ..
!make -j8
!make install
```

```
-- Up-to-date: /usr/local/lib/python3.10/dist-packages/cv2/motempl
-- Installing: /usr/local/lib/python3.10/dist-packages/cv2/motempl/__init__.pyi
-- Up-to-date: /usr/local/lib/python3.10/dist-packages/cv2/bgsegm
-- Installing: /usr/local/lib/python3.10/dist-packages/cv2/bgsegm/__init__.pyi
-- Up-to-date: /usr/local/lib/python3.10/dist-packages/cv2/typing
-- Installing: /usr/local/lib/python3.10/dist-packages/cv2/typing/__init__.py
-- Up-to-date: /usr/local/lib/python3.10/dist-packages/cv2/kinfu
-- Up-to-date: /usr/local/lib/python3.10/dist-packages/cv2/kinfu/detail
-- Installing: /usr/local/lib/python3.10/dist-packages/cv2/kinfu/detail/__init__.pyi
-- Installing: /usr/local/lib/python3.10/dist-packages/cv2/kinfu/__init__.pyi
-- Up-to-date: /usr/local/lib/python3.10/dist-packages/cv2/rapid
-- Installing: /usr/local/lib/python3.10/dist-packages/cv2/rapid/__init__.pyi
-- Up-to-date: /usr/local/lib/python3.10/dist-packages/cv2/img_hash
-- Installing: /usr/local/lib/python3.10/dist-packages/cv2/img_hash/__init__.pyi
-- Up-to-date: /usr/local/lib/python3.10/dist-packages/cv2/omnidir
-- Installing: /usr/local/lib/python3.10/dist-packages/cv2/omnidir/__init__.pyi
-- Up-to-date: /usr/local/lib/python3.10/dist-packages/cv2/parallel
-- Installing: /usr/local/lib/python3.10/dist-packages/cv2/parallel/__init__.pyi
-- Up-to-date: /usr/local/lib/python3.10/dist-packages/cv2/saliency
-- Installing: /usr/local/lib/python3.10/dist-packages/cv2/saliency/__init__.pyi
-- Up-to-date: /usr/local/lib/python3.10/dist-packages/cv2/signal
-- Installing: /usr/local/lib/python3.10/dist-packages/cv2/signal/__init__.pyi
-- Installing: /usr/local/lib/python3.10/dist-packages/cv2/python-3.10/cv2.cpython-310-x86_64-linux-gnu.so
-- Set non-toolchain portion of runtime path of "/usr/local/lib/python3.10/dist-packages/cv2/python-3.10/cv2.cpython-310-x86_64-linux
-- Installing: /usr/local/lib/python3.10/dist-packages/cv2/config-3.10.py
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_eye.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_eye_tree_eyeglasses.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_frontalcatface.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_frontalcatface_extended.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_frontalface_alt.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_frontalface_alt2.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_frontalface_alt_tree.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_frontalface_default.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_fullbody.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_lefteye_2splits.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_license_plate_rus_16stages.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_lowerbody.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_profileface.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_righteye_2splits.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_russian_plate_number.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_smile.xml
-- Installing: /usr/local/share/opencv4/haarcascades/haarcascade_upperbody.xml
-- Installing: /usr/local/share/opencv4/lbpcascades/lbpcascade_frontalcatface.xml
-- Installing: /usr/local/share/opencv4/lbpcascades/lbpcascade_frontalface.xml
-- Installing: /usr/local/share/opencv4/lbpcascades/lbpcascade_frontalface_improved.xml
-- Installing: /usr/local/share/opencv4/lbpcascades/lbpcascade_profileface.xml
-- Installing: /usr/local/share/opencv4/lbpcascades/lbpcascade_silverware.xml
-- Installing: /usr/local/bin/opencv_annotation
-- Set non-toolchain portion of runtime path of "/usr/local/bin/opencv_annotation" to "/usr/local/lib"
-- Installing: /usr/local/bin/opencv_visualisation
-- Set non-toolchain portion of runtime path of "/usr/local/bin/opencv_visualisation" to "/usr/local/lib"
-- Installing: /usr/local/bin/opencv_interactive-calibration
-- Set non-toolchain portion of runtime path of "/usr/local/bin/opencv_interactive-calibration" to "/usr/local/lib"
-- Installing: /usr/local/bin/opencv_version
-- Set non-toolchain portion of runtime path of "/usr/local/bin/opencv_version" to "/usr/local/lib"
-- Installing: /usr/local/bin/opencv_model_diagnostics
-- Set non-toolchain portion of runtime path of "/usr/local/bin/opencv_model_diagnostics" to "/usr/local/lib"
```

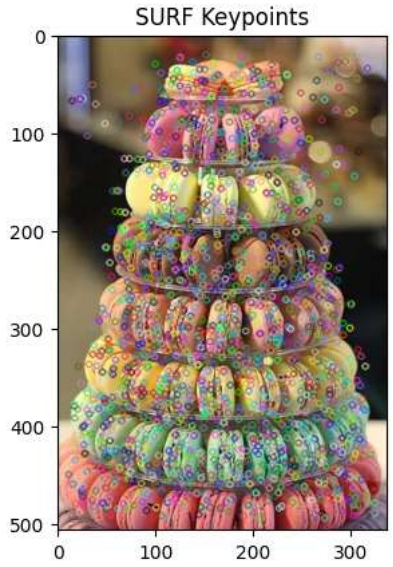
## Task 1 SIFT Feature Extraction

```
import cv2
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('/content/img.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Initialize SURF detector
surf = cv2.xfeatures2d.SURF_create()

# Detect keypoints and descriptors
keypoints, descriptors = surf.detectAndCompute(gray_image, None)

# Draw keypoints on the image
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None)

# Display the image with keypoints
plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB))
plt.title('SURF Keypoints')
plt.show()
```




## Task 2 Surf Feature Extraction

```
import cv2
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('/content/img.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Initialize SURF detector (requires OpenCV contrib package with non-free enabled)
surf = cv2.xfeatures2d.SURF_create()

# Detect keypoints and descriptors
keypoints, descriptors = surf.detectAndCompute(gray_image, None)

# Draw keypoints on the image
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None)

# Display the image with keypoints
plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB))
plt.title('SURF Keypoints')
plt.show()
```

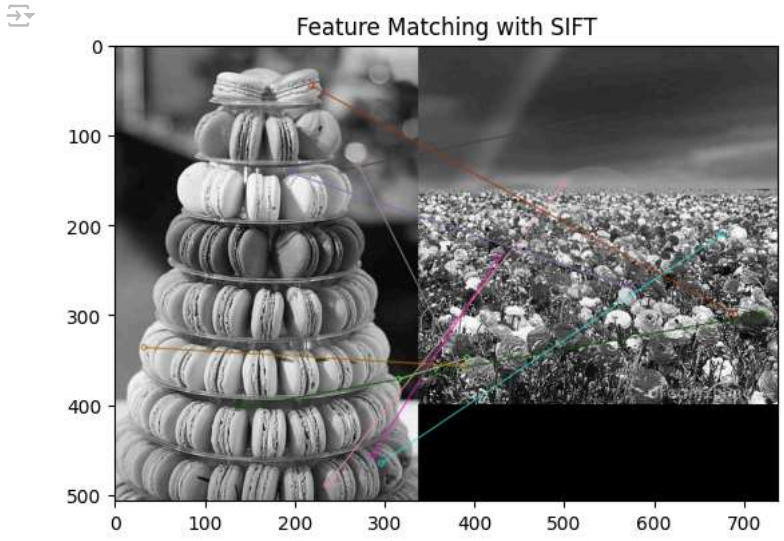



## Task 3 ORB Feature Extraction

```
# Read the image
image = cv2.imread("/content/img.jpg") # Added /content to file path

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Initialize the ORB detector
orb = cv2.ORB_create()

# Detect keypoints and descriptors
keypoints, descriptors = orb.detectAndCompute(gray_image, None)

# Draw keypoints on the image
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None)

# Display the result
plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB))
plt.title("ORB KEYPOINTS")
plt.show()
```


## Task 4 Feature Matching

```
# Load two images
image1 = cv2.imread('/content/img.jpg', 0)
image2 = cv2.imread('/content/img2.jpg', 0)

# Initialize SIFT detector
sift = cv2.SIFT_create()

# Find keypoints and descriptors with SIFT
keypoints1, descriptors1 = sift.detectAndCompute(image1, None)
keypoints2, descriptors2 = sift.detectAndCompute(image2, None)

# Initialize the matcher
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)

# Match descriptors
matches = bf.match(descriptors1, descriptors2)

# Sort matches by distance (best matches first)
matches = sorted(matches, key=lambda x: x.distance)

# Draw matches
image_matches = cv2.drawMatches(image1, keypoints1, image2, keypoints2, matches[:10], None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINT

# Display the matches
plt.imshow(image_matches)
plt.title('Feature Matching with SIFT')
plt.show()
```




## *Task 5 Applications of Feature Matching*

```
# Load two images
image1 = cv2.imread('/content/img.jpg', cv2.IMREAD_GRAYSCALE)
image2 = cv2.imread('/content/img2.jpg', cv2.IMREAD_GRAYSCALE)

# Detect keypoints and descriptors using SIFT
sift = cv2.SIFT_create()
keypoints1, descriptors1 = sift.detectAndCompute(image1, None)
keypoints2, descriptors2 = sift.detectAndCompute(image2, None)

# Match features using BFMatcher
bf = cv2.BFMatcher(cv2.NORM_L2)
matches = bf.knnMatch(descriptors1, descriptors2, k=2)

# Apply ratio test (Lowe's ratio test)
good_matches = []
for m, n in matches:
    if m.distance < 0.75 * n.distance:
        good_matches.append(m)

# Extract location of good matches
```

```
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)

# Find homography matrix
M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

# Check if a valid homography matrix was found
if M is not None:
    # Warp one image to align with the other
    h, w = image1.shape
    result = cv2.warpPerspective(image1, M, (w, h))

    # Display the result
    plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
    plt.title('Image Alignment using Homography')
    plt.show()
else:
    print("Homography matrix could not be found.")
```

```
Homography matrix could not be found.
```

## Combining Feature Extraction Methods

```
import cv2

# Use SIFT and ORB to extract features from two images

# Load two images
# Ensure correct paths to images
image1 = cv2.imread('/content/img.jpg', 0)
image2 = cv2.imread('/content/img2.jpg', 0)

# SIFT detector
sift = cv2.SIFT_create()
keypoints1_sift, descriptors1_sift = sift.detectAndCompute(image1, None)
keypoints2_sift, descriptors2_sift = sift.detectAndCompute(image2, None)

# ORB detector
orb = cv2.ORB_create()
keypoints1_orb, descriptors1_orb = orb.detectAndCompute(image1, None)
keypoints2_orb, descriptors2_orb = orb.detectAndCompute(image2, None)
```

## Processed Images

```
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
import cv2
import numpy as np

# Assuming you have the following variables from your previous code:
# image_with_keypoints (SIFT), image_with_keypoints (SURF), image_with_keypoints (ORB), image_matches (feature matching)

# Load two images
image1 = cv2.imread('/content/img.jpg', cv2.IMREAD_GRAYSCALE)
image2 = cv2.imread('/content/img2.jpg', cv2.IMREAD_GRAYSCALE)

# Detect keypoints and descriptors using SIFT
sift = cv2.SIFT_create()
keypoints1, descriptors1 = sift.detectAndCompute(image1, None)
keypoints2, descriptors2 = sift.detectAndCompute(image2, None)

# Match features using BFMatcher
bf = cv2.BFMatcher(cv2.NORM_L2)
matches = bf.knnMatch(descriptors1, descriptors2, k=2)

# Apply ratio test (Lowe's ratio test)
good_matches = []
for m, n in matches:
    if m.distance < 0.75 * n.distance:
        good_matches.append(m)
```

```
# Extract location of good matches
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)

# Find homography matrix
M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

# Check if a valid homography matrix was found
if M is not None:
    # Warp one image to align with the other
    h, w = image1.shape
    result = cv2.warpPerspective(image1, M, (w, h))

with PdfPages('processed_images.pdf') as pdf:
    plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB))
    plt.title("SIFT Keypoints")
    pdf.savefig()
    plt.close()

    plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB))  # Replace with your SURF image
    plt.title("SURF Keypoints")
    pdf.savefig()
    plt.close()

    plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB))  # Replace with your ORB image
    plt.title("ORB Keypoints")
    pdf.savefig()
    plt.close()

    plt.imshow(image_matches)
    plt.title("Feature Matching with SIFT")
    pdf.savefig()
    plt.close()

    # ensure that result is defined
    if M is not None:
        plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
        plt.title("Image Alignment using Homography")
        pdf.savefig()
        plt.close()


print("PDF saved as Exer2.pdf")
```

```
PDF saved as Exer2.pdf
```