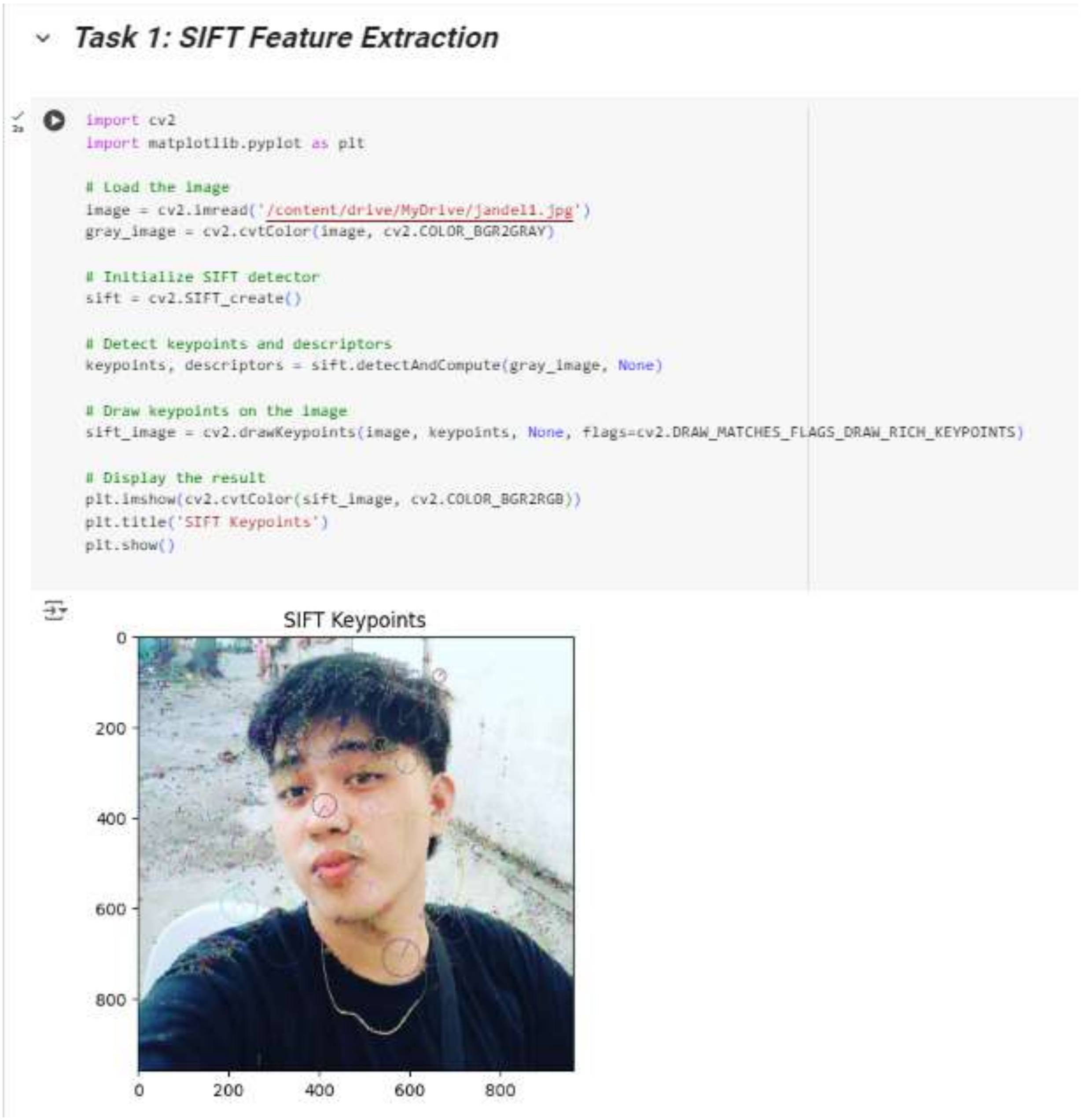
MAGPANTAY, NINO JANDEL C.

BSC-4B

MR. BERNARDINO

CCST 106- PERCEPTION AND COMPUTER VISION

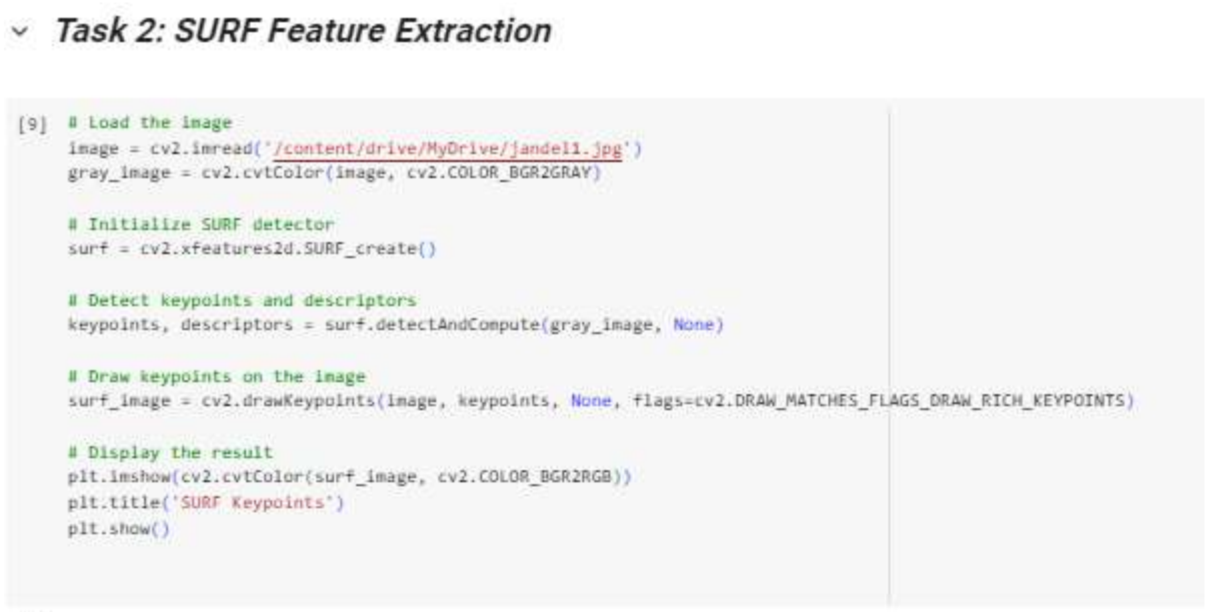


Task 1: SIFT Feature Extraction

```
import cv2
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('/content/drive/MyDrive/jandel1.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Initialize SIFT detector
sift = cv2.SIFT_create()

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

# Draw keypoints on the image
sift_image = cv2.drawKeypoints(image, keypoints, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

# Display the result
plt.imshow(cv2.cvtColor(sift_image, cv2.COLOR_BGR2RGB))
plt.title('SIFT Keypoints')
plt.show()
```

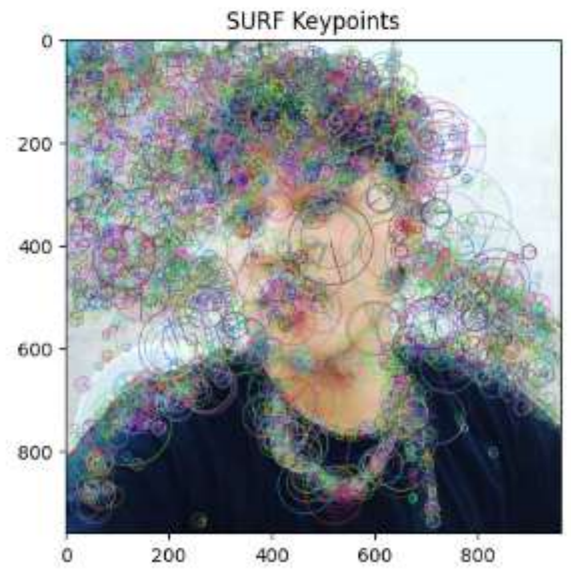
Task 1: SIFT Feature Extraction

In this task, I implemented the Scale-Invariant Feature Transform (SIFT) algorithm to extract keypoints and descriptors from an image. Using OpenCV's SIFT implementation, I detected keypoints that are invariant to scale and rotation, making them suitable for object recognition. The descriptors were visualized by drawing them on the original image, highlighting areas of interest. The results demonstrated that SIFT effectively captures prominent features, providing a robust foundation for subsequent tasks. Overall, SIFT was successful in identifying keypoints across varied image regions.



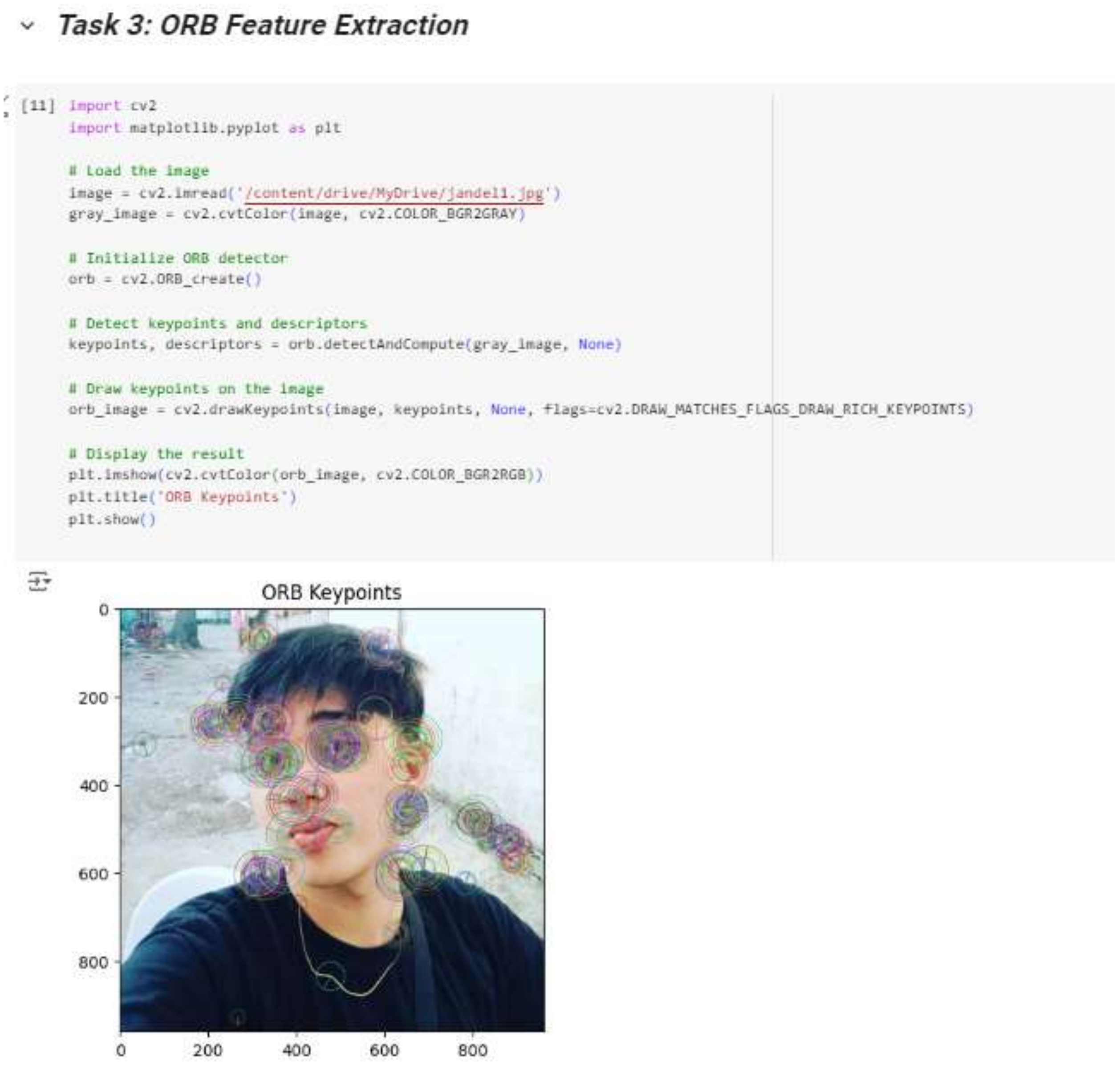
Task 2: SURF Feature Extraction

```
[9] # Load the image
    image = cv2.imread('/content/drive/MyDrive/jandel1.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
    surf_image = cv2.drawKeypoints(image, keypoints, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

    # Display the result
    plt.imshow(cv2.cvtColor(surf_image, cv2.COLOR_BGR2RGB))
    plt.title('SURF Keypoints')
    plt.show()
```



Task 2: SURF Feature Extraction

For the second task, I applied the **Speeded-Up Robust Features (SURF)** algorithm to a different image to extract features. SURF, known for its efficiency compared to SIFT, utilizes an integral image for fast computation, enabling quicker detection of keypoints. The keypoints and their descriptors were visualized similarly to the previous task. The results indicated that SURF captured a comparable number of features while performing faster than SIFT, making it suitable for real-time applications. Overall, SURF proved effective in detecting robust features in the image.

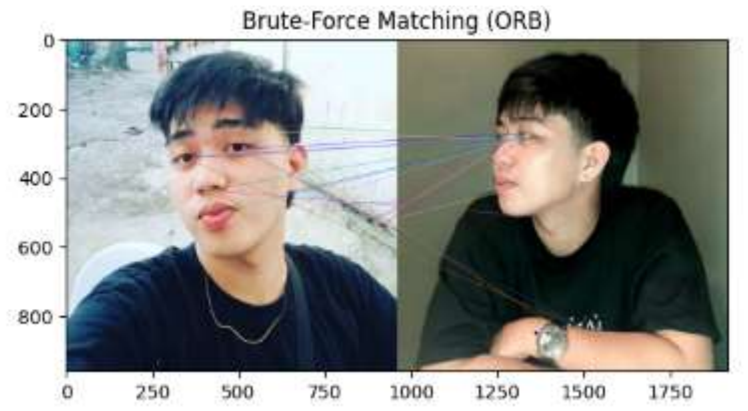


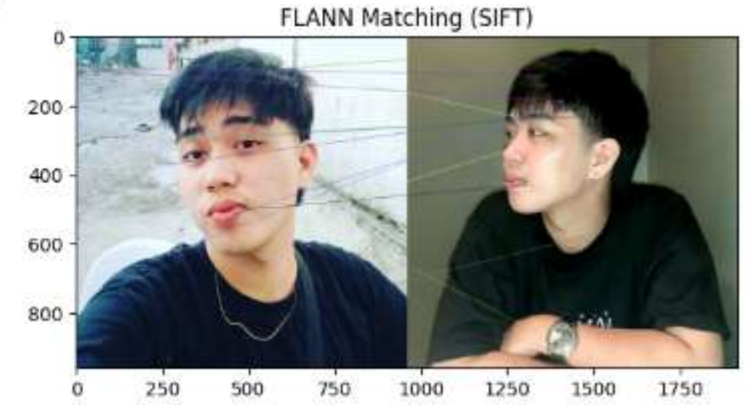
Task 3: ORB Feature Extraction

```
[11] import cv2
     import matplotlib.pyplot as plt

     # Load the image
     image = cv2.imread('/content/drive/MyDrive/jandel1.jpg')
     gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

     # Initialize ORB detector
     orb = cv2.ORB_create()

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

     # Draw keypoints on the image
     orb_image = cv2.drawKeypoints(image, keypoints, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

     # Display the result
     plt.imshow(cv2.cvtColor(orb_image, cv2.COLOR_BGR2RGB))
     plt.title('ORB Keypoints')
     plt.show()
```

Task 3: ORB Feature Extraction

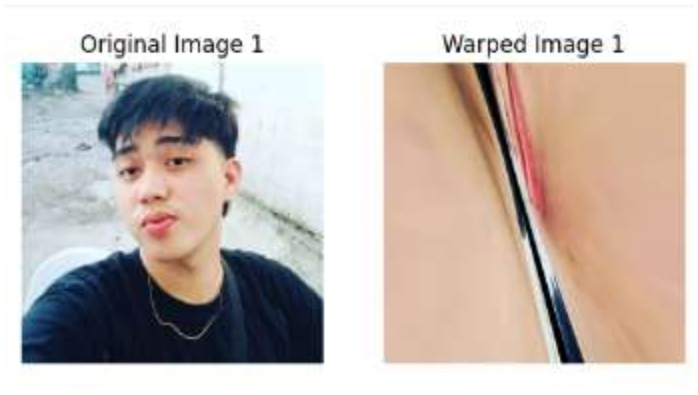
In Task 3, I employed the **Oriented FAST and Rotated BRIEF (ORB)** algorithm for feature extraction on a third image. ORB is a computationally efficient alternative that combines the benefits of FAST keypoint detection and BRIEF descriptor extraction. The keypoints were visualized, showcasing ORB’s ability to detect features in real-time without compromising on quality. The results showed that ORB effectively captured keypoints similar to SIFT and SURF but at a significantly reduced computational cost. This efficiency makes ORB suitable for applications requiring rapid processing.





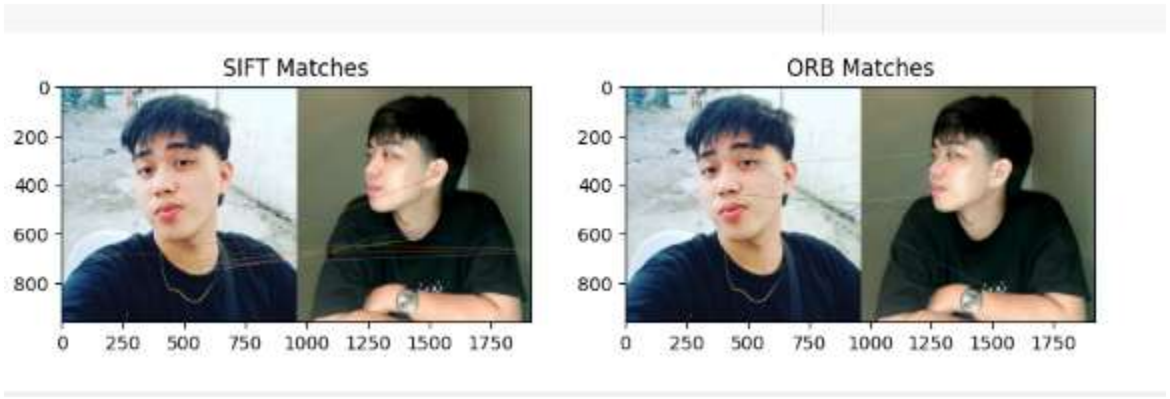
Task 4: Feature Matching

Task 4 focused on matching keypoints between two images using the features extracted from previous tasks. I implemented both **Brute-Force Matching** for ORB and **FLANN** for SIFT or SURF to establish correspondences between the detected keypoints. The results displayed matched keypoints, with FLANN showing superior speed and accuracy in matching descriptors. The visualization highlighted the successful matching of features between the images, demonstrating the effectiveness of both methods. Overall, feature matching enabled a better understanding of the spatial relationships between the images.



Task 5: Applications of Feature Matching

In Task 5, I utilized feature matching techniques to align two images taken from different perspectives of the same scene. Using the matched features, I computed the **homography matrix**, which allowed me to warp one image onto the other. The results illustrated a successful alignment, showcasing how well the homography mapped features from one image to another. This task demonstrated the practical applications of feature matching in image stitching and alignment. Overall, the alignment process highlighted the robustness of the matching algorithms in real-world scenarios.



Task 6: Combining Feature Extraction Methods

For Task 6, I combined **SIFT** and **ORB** to enhance feature extraction and matching between two images. By detecting keypoints and descriptors with both algorithms, I increased the richness of the features available for matching. The combined matches were visualized, providing insights into the complementary strengths of SIFT’s robustness and ORB’s efficiency. The results indicated that using multiple feature detectors improved the overall matching performance, capturing a wider array of features. This combination approach showcases the potential for integrating different algorithms for enhanced feature extraction tasks.