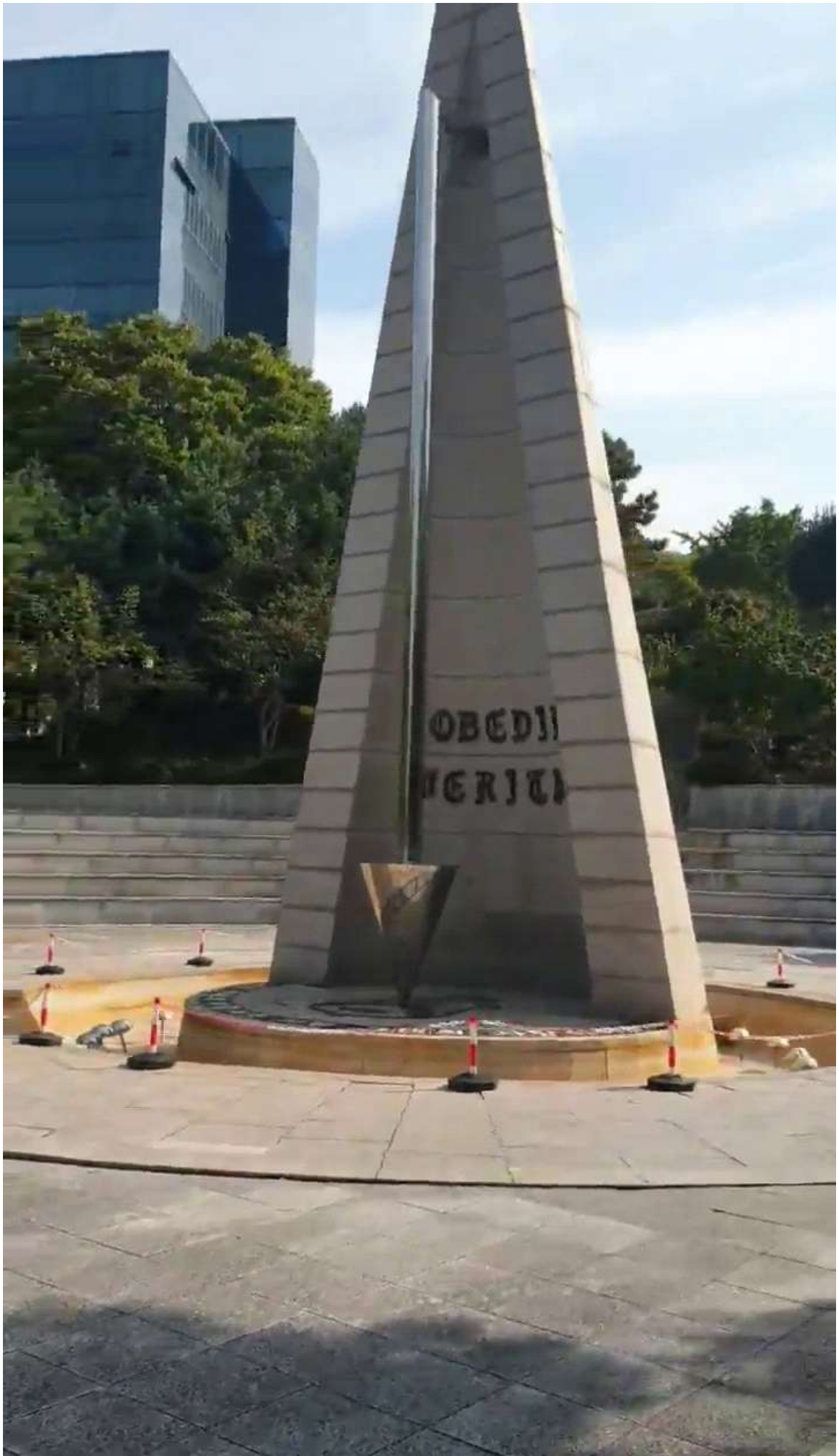
120230225

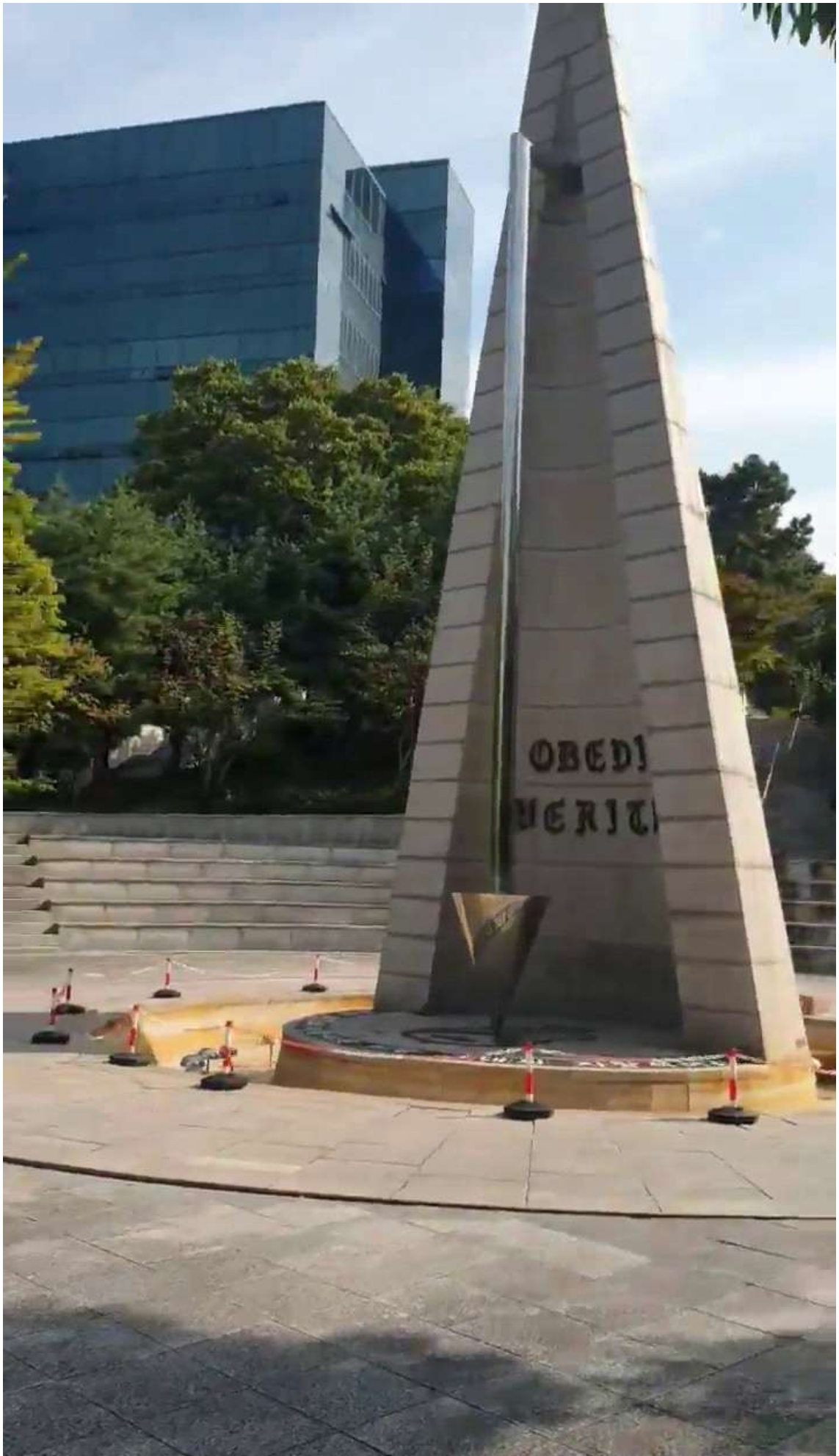
비주얼오도메트리와증강현실

과제

강찬희

1. Choose two images


OBEDII
ERITI


OBEDI
VERITI

2. Compute ORB keypoint and descriptors and apply bruteforce matching with Hamming distance (opencv)

```python
def ORB_matching(img1, img2):
    # Initialize ORB detector
    orb = cv2.ORB_create()

    # Find the keypoints and descriptors with ORB
    kp1, des1 = orb.detectAndCompute(img1, None)
    kp2, des2 = orb.detectAndCompute(img2, None)

    # Create BFMatcher object with Hamming distance and crossCheck is turned on for better matching
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

    # Match descriptors
    matches = bf.match(des1, des2)

    # Sort them in the order of distance
    matches = sorted(matches, key=lambda x: x.distance)

    return kp1, kp2, matches
```

Use ORB detector from opencv-python library

BFMatcher stands for BruteForcaeMatcher

And then sort them in the order of distance (hamming distance)

3. Implement RANSAC algorithm to compute the homography matrix (DIY)

```python
def compute_homography(src_pts, dst_pts):
    A = []

    for i in range(0, len(src_pts)):
        x, y = src_pts[i]
        u, v = dst_pts[i]
        A.append([-x, -y, -1, 0, 0, 0, u*x, u*y, u])
        A.append([0, 0, 0, -x, -y, -1, v*x, v*y, v])

    A = np.asarray(A)
    U, S, Vh = np.linalg.svd(A)
    L = Vh[-1,:] / Vh[-1,-1]
    H = L.reshape(3, 3)

    return H
```

```python
def custom_RANSAC(kp1, kp2, matches, iterations=1000, tolerance=5):
    best_H = None
    max_inliers = 0

    for i in range(iterations):
        # Randomly select 4 matches
        selected_matches = np.random.choice(matches, 4, replace=False)

        src_points = np.float32([kp1[m.queryIdx].pt for m in selected_matches])
        dst_points = np.float32([kp2[m.trainIdx].pt for m in selected_matches])

        # Compute homography from 4 matches
        H = compute_homography(src_points, dst_points)

        inliers_count = 0

        for m in matches:
            p1 = np.array([kp1[m.queryIdx].pt[0], kp1[m.queryIdx].pt[1], 1])
            estimated_p2 = np.dot(H, p1)
            estimated_p2 = estimated_p2/estimated_p2[-1]

            p2 = np.array([kp2[m.trainIdx].pt[0], kp2[m.trainIdx].pt[1], 1])

            error = np.linalg.norm(p2 - estimated_p2)

            if error < tolerance:
                inliers_count += 1

        if inliers_count > max_inliers:
            max_inliers = inliers_count
            best_H = H

    return best_H
```

Compute_homography() is a function that calculates the homography matrix matched point pairs. It constructs a matrix by combining the coordinates of matched points and then applies SVD to solve for the homography matrix.

Since the achieving a good homography matrix highly denpends on the quality of the matched points, it might require fine-tuning parameters such as RANSAC iterations and the error tolerance.

4. Prepare a panorama image of larger size and warp two images to the panorama image using the homography matrix (DIY)

```python
def warp_images_custom(img1, img2, H):
    h1, w1 = img1.shape[:2]
    h2, w2 = img2.shape[:2]

    # Find corner points and apply perspective transformation
    corners = [[0, 0], [0, h1-1], [w1-1, h1-1], [w1-1, 0]]
    transformed_corners = [np.dot(H, [x, y, 1]) for (x, y) in corners]
    transformed_corners = [[x[0]/x[2], x[1]/x[2]] for x in transformed_corners]

    # Find the size of the new image
    min_x = min(transformed_corners[0][0], transformed_corners[1][0], 0, w2)
    min_y = min(transformed_corners[0][1], transformed_corners[3][1], 0, h2)
    max_x = max(transformed_corners[2][0], transformed_corners[3][0], w1, w2)
    max_y = max(transformed_corners[1][1], transformed_corners[2][1], h1, h2)

    offset_x = 0 if min_x > 0 else -min_x
    offset_y = 0 if min_y > 0 else -min_y

    # Create the new image (panorama)
    panorama = np.zeros((int(max_y-min_y)+1, int(max_x-min_x)+1, 3), dtype=np.uint8)

    # Paste the first image
    panorama[int(offset_y):int(offset_y)+h1, int(offset_x):int(offset_x)+w1] = img1

    # Paste the second image
    for y in range(h2):
        for x in range(w2):
            p = np.dot(np.linalg.inv(H), [x, y, 1])
            p = [p[0]/p[2], p[1]/p[2]]

            if 0 ≤ p[1] < h1 and 0 ≤ p[0] < w1:
                panorama[int(p[1]+offset_y), int(p[0]+offset_x)] = img2[y, x]

    return panorama
```

```python
# Load images
img1 = cv2.imread('dataset/I/alba1.png')
img2 = cv2.imread('dataset/I/alba2.png')

# Compute ORB keypoints and descriptors, and perform Bruteforce matching
kp1, kp2, matches = ORB_matching(img1, img2)

# Compute the homography matrix using a custom RANSAC implementation
H = custom_RANSAC(kp1, kp2, matches)

# Prepare the panorama image by warping two images using the homography matrix
panorama = warp_images_custom(img1, img2, H)

# Save or display the panorama image
cv2.imwrite('panorama.jpg', panorama)
```

5. Final Result

```python
import matplotlib.pyplot as plt

panorama = cv2.cvtColor(panorama,cv2.COLOR_BGR2RGB)

plt.imshow(panorama)
plt.show()
```

✓ 0.2s