Lab-05 Report

st122246

July 15, 2022

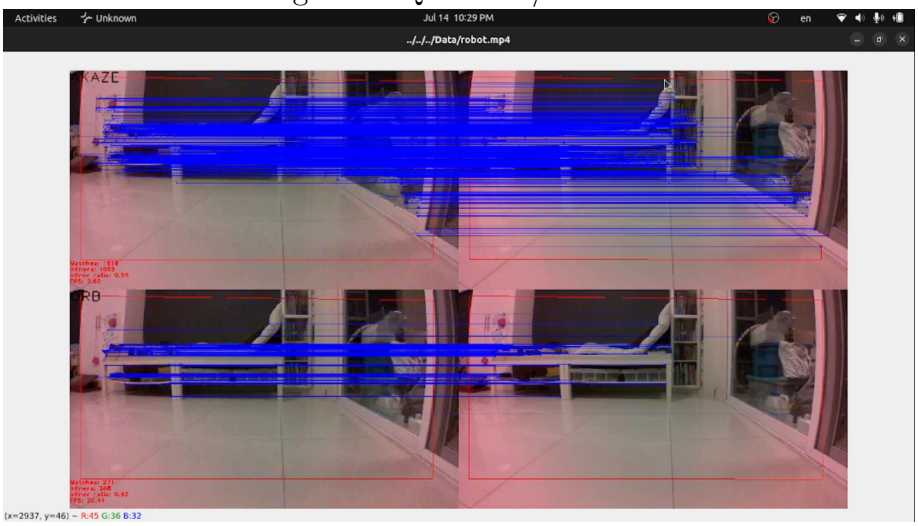
1 Q1: ORB/AKAZE Tutorial

2 Q2: Feature Points

Detect feature points

```
1 # ---------- Create detector ----------
2 detector1 = cv.AKAZE_create()
3 detector2 = cv.ORB_create()
4
5 def detect_key_points(self):
6     """
7     Detect features in two frames.
8     """
9     img1_kps, img1_desc = self.detector.
    detectAndCompute(self.img1, None)
10    img2_kps, img2_desc = self.detector.
    detectAndCompute(self.img2, None)
11
12    res_img1 = cv.drawKeypoints(self.img1, img1_kps,
    None, (255, 0, 0), cv.
    DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
13    res_img2 = cv.drawKeypoints(self.img2, img1_kps,
    None, (255, 0, 0), cv.
    DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
14
15    res_img1 = cv.putText(res_img1, "Image 1", (0, 60),
     cv.FONT_HERSHEY_PLAIN, 3, (0, 0, 255), 2)
16    res_img2 = cv.putText(res_img2, "Image 2", (0, 60),
     cv.FONT_HERSHEY_PLAIN, 3, (0, 0, 255), 2)
```

Figure 1: Q1: ORB/AKAZE Tutorial

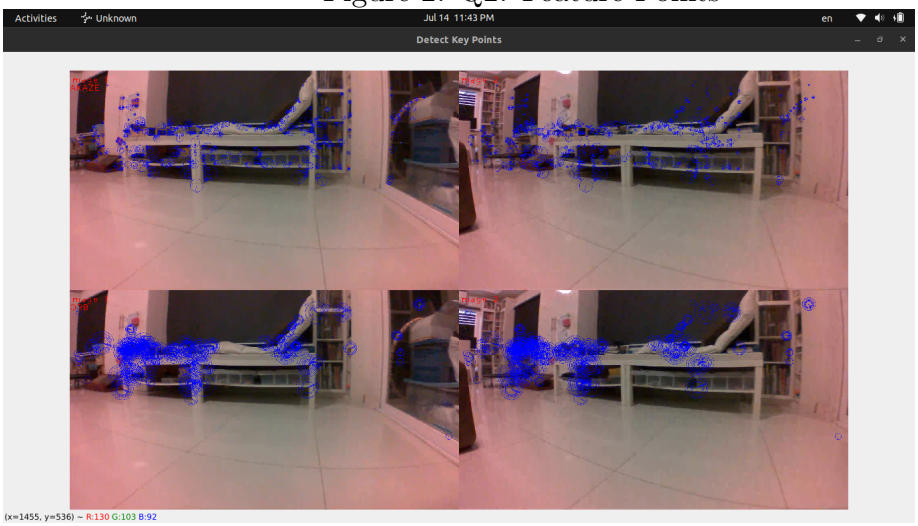



```
17
18      res_img = cv.hconcat([res_img1, res_img2])
19
20      self._img1_kps = img1_kps
21      self._img1_desc = img1_desc
22      self._img2_kps = img2_kps
23      self._img2_desc = img2_desc
24
25      return res_img, img1_kps, img1_desc, img2_kps,
    img2_desc
26
```

3 Q3: Undistortion

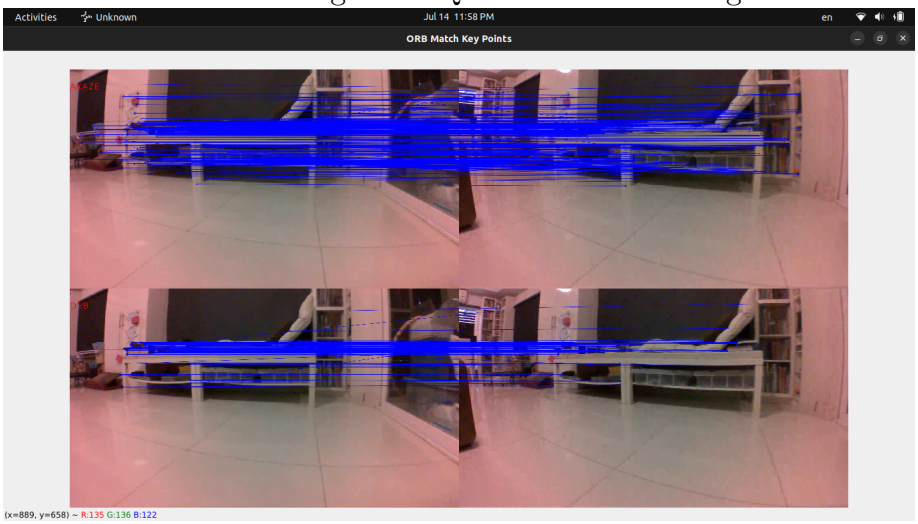
```
1 def _undistorted_image(self, image):
2      h, w = image.shape[:2]
3      new_camera_mtx, roi = cv.getOptimalNewCameraMatrix(
    self.camera_mtx, self.camera_dist, (w, h), 1, (w, h)
    )
4      undistorted_img = cv.undistort(image, self.
    camera_mtx, self.camera_dist, None, new_camera_mtx)
5      x, y, w, h = roi
6      undistorted_img = undistorted_img[y:y+h, x:x+w]
```

Figure 2: Q2: Feature Points




```
7       return undistorted_img, new_camera_mtx, x, y
8
9   def _undistorted_points(self, points, new_camera_matrix
        , new_x, new_y):
10      undistorted_pts = cv.undistortPoints(points, self.
        camera_mtx, self.camera_dist, P=new_camera_matrix)
11      undistorted_pts = np.array([[pt[0, 0]-new_x, pt[0,
        1]-new_y] for pt in undistorted_pts], dtype=np.
        float32)
12      return undistorted_pts
13
14  def _undistorted_key_points(self, key_points,
        new_camera_matrix, new_x, new_y):
15      new_key_points = list()
16
17      pts = np.array([keypoint.pt for keypoint in
        key_points], dtype=np.float32)
18      undistorted_pts = self._undistorted_points(pts,
        new_camera_matrix, new_x, new_y)
19
20      for i, (kp, pt) in enumerate(zip(key_points,
        undistorted_pts)):
21          new_key_point = cv.KeyPoint(x=pt[0], y=pt[1],
        size=kp.size, angle=kp.angle, response=kp.response,
```

Figure 3: Q4: Feature Matching

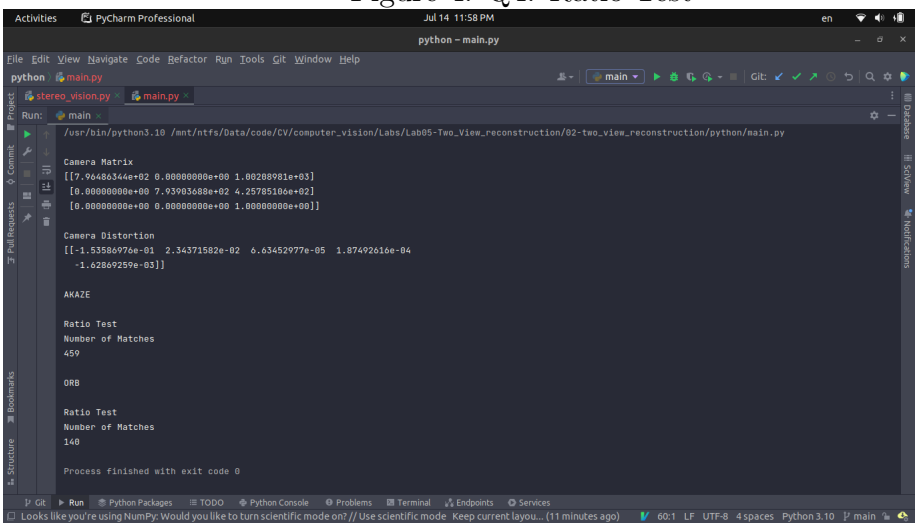



```
22                                                    octave=kp.octave,
    class_id=kp.class_id)
23             new_key_points.append(new_key_point)
24         new_key_points = tuple(new_key_points)
25         return new_key_points
26
```

4 Q4: Feature Matching

```
1 # ---------- Match Key Points ----------
2 nn_match_ratio: float = 0.8
3 print("\nAKAZE")
4 akaze_matched_img, akaze_pts1, akaze_pts2,
    akaze_good_matches = stereo_vision1.match_key_points
    (nn_match_ratio)
5 print("\nORB")
6 orb_matched_img, orb_pts1, orb_pts2, orb_good_matches =
     stereo_vision2.match_key_points(nn_match_ratio)
7 cv.putText(akaze_matched_img, "AKAZE", (0, 100), cv.
    FONT_HERSHEY_PLAIN, 3, (0, 0, 255), 2)
8 cv.putText(orb_matched_img, "ORB", (0, 100), cv.
    FONT_HERSHEY_PLAIN, 3, (0, 0, 255), 2)
9 img = cv.vconcat([akaze_matched_img, orb_matched_img])
```

Figure 4: Q4: Ratio Test




```
10 show_result_img("ORB Match Key Points", img)
11
12 def match_key_points(self, match_ratio=0.8):
13     matcher = cv.DescriptorMatcher_create("BruteForce-
       Hamming")
14
15     matches = matcher.knnMatch(self._img1_desc, self.
       _img2_desc, k=2)
16
17     good_matches = list()
18     for i, (m, n) in enumerate(matches):
19         if m.distance < match_ratio * n.distance:
20             good_matches.append(m)
21     pts1 = np.array([self._img1_kps[m.queryIdx].pt for
       m in good_matches], dtype=np.float32)
22     pts2 = np.array([self._img2_kps[m.trainIdx].pt for
       m in good_matches], dtype=np.float32)
23
24     print("\nRatio Test")
25     print("Number of Matches")
26     print(pts1.shape[0])
27
28     res_img = cv.drawMatches(self.img1, self._img1_kps,
       self.img2, self._img2_kps, good_matches, None,
```

Figure 5: Q5: Matching Using Essential Matrix

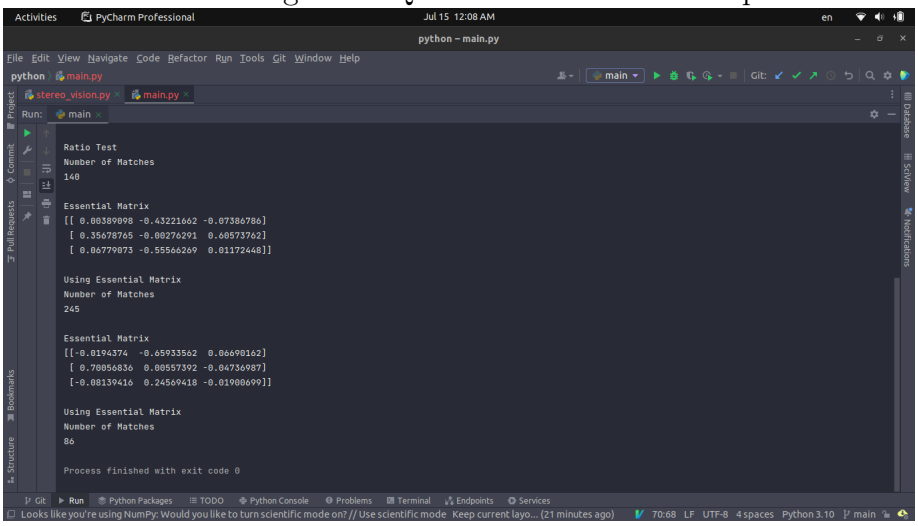

```
    (255, 0, 0),
29                                  (255, 0, 0), flags=2)
30
31      self._pts1 = pts1
32      self._pts2 = pts2
33      self._good_matches = good_matches
34      return res_img, pts1, pts2, good_matches
35
```

5 Q5: Essential Matrix

Find essential matrix and use it to get better correspondences.

```
1   # ---------- Find Essential Matrix ----------
2 akaze_essential_img, akaze_essential_matrix,
      akaze_essential_pts1, akaze_essential_pts2,
      akaze_new_good_matches = \
3       stereo_vision1.find_essential_matrix(akaze_pts1,
      akaze_pts2)
4 orb_essential_img, orb_essential_matrix,
      orb_essential_pts1, orb_essential_pts2,
      orb_new_good_matches = \
5       stereo_vision2.find_essential_matrix(orb_pts1,
      orb_pts2)
```

Figure 6: Q5: Essential Matrix Output



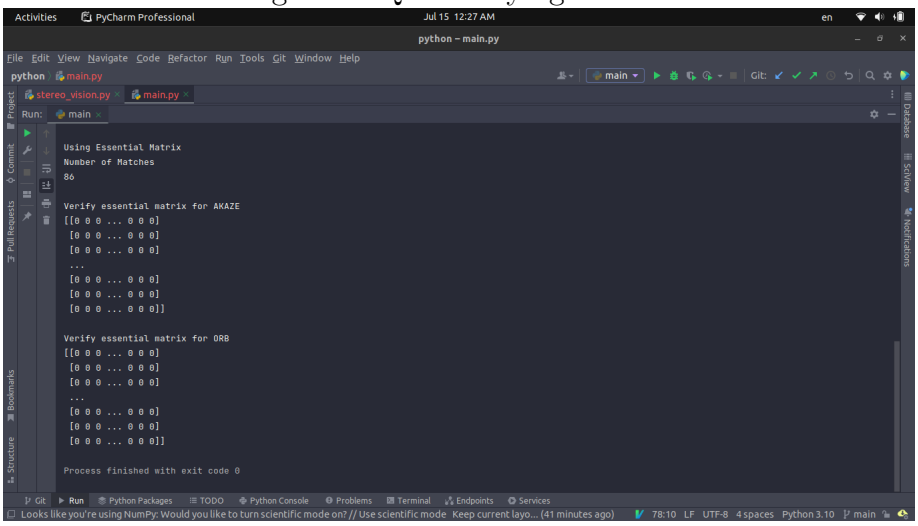

```
6
7 cv.putText(akaze_essential_img, "AKAZE", (0, 100), cv.
    FONT_HERSHEY_PLAIN, 3, (0, 0, 255), 2)
8 cv.putText(orb_essential_img, "ORB", (0, 100), cv.
    FONT_HERSHEY_PLAIN, 3, (0, 0, 255), 2)
9 img = cv.vconcat([akaze_essential_img,
    orb_essential_img])
10 show_result_img("ORB Match Key Points Using Essential
    Matrix", img)
11
12 def find_essential_matrix(self, points1, points2):
13     essential_matrix, mask = cv.findEssentialMat(
    points1, points2, self.camera_mtx, cv.FM_RANSAC)
14     print("\nEssential Matrix")
15     print(essential_matrix)
16     pts1 = points1[mask.ravel() == 1]
17     pts2 = points2[mask.ravel() == 1]
18     new_good_matches = [match for match, m in zip(self.
    _good_matches, mask.ravel() == 1) if m]
19
20     print("\nUsing Essential Matrix")
21     print("Number of Matches")
22     print(len(new_good_matches))
23
```

```
24      res_img = cv.drawMatches(self.img1, self._img1_kps,
        self.img2, self._img2_kps, new_good_matches, None,
25                               (255, 0, 0), (255, 0, 0),
    flags=2)
26
27      self.essential_matrix = essential_matrix
28      self._essential_pts1 = pts1
29      self._essential_pts2 = pts2
30      self._essential_matches = new_good_matches
31
32      return res_img, essential_matrix, pts1, pts2,
    new_good_matches
33
```

Verify essential matrix

```
1 # ---------- Verify Essential Matrix ----------
2 print("\nVerify essential matrix for AKAZE")
3 akaze_result = stereo_vision1.verify_essential_matrix(
    akaze_essential_pts1, akaze_essential_pts2,
    akaze_essential_matrix)
4 print(akaze_result)
5 print("\nVerify essential matrix for ORB")
6 orb_result = stereo_vision1.verify_essential_matrix(
    orb_essential_pts1, orb_essential_pts2,
    orb_essential_matrix)
7 print(orb_result)
8
9 def verify_essential_matrix(self, points1, points2,
    essential_matrix):
10      x = np.concatenate((points1, np.ones((points1.shape
    [0], 1))), axis=1).T
11      x_prime = np.concatenate((points2, np.ones((points2
    .shape[0], 1))), axis=1).T
12      k = self.camera_mtx
13      k_inv = np.linalg.inv(k)
14      e = essential_matrix
15      result = x.T @ k_inv.T @ e @ k_inv @ x_prime
16      result = result.astype(np.int32)
17      return result
18
```

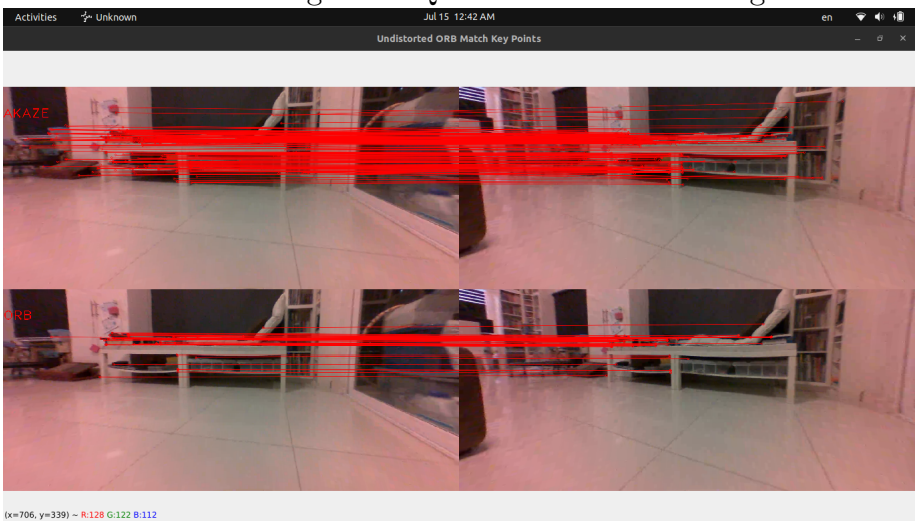
Figure 7: Q5: Verifying Essential Matrix



6 Q6: Undistored Matching

```
1 # ------------ Undistorted Matches ------------
2 akaze_undistorted_matches = stereo_vision1.
      undistorted_matches(left_img, right_img,
      akaze_img1_kps, akaze_img2_kps,
      akaze_new_good_matches)
3 orb_undistorted_matches = stereo_vision2.
      undistorted_matches(left_img, right_img,
      orb_img1_kps, orb_img2_kps, orb_new_good_matches)
4
5 cv.putText(akaze_undistorted_matches, "AKAZE", (0, 100)
      , cv.FONT_HERSHEY_PLAIN, 3, (0, 0, 255), 2)
6 cv.putText(orb_undistorted_matches, "ORB", (0, 100), cv
      .FONT_HERSHEY_PLAIN, 3, (0, 0, 255), 2)
7 img = cv.vconcat([akaze_undistorted_matches,
      orb_undistorted_matches])
8 show_result_img("Undistorted ORB Match Key Points", img
      )
9
10 def undistorted_matches(self, image1, image2,
      image1_kps, image2_kps, good_matches):
11     undistorted_img1, img1_new_cam_mtx, img1_x, img1_y
      = self._undistorted_image(image1)
```

Figure 8: Q6: Undistored Matching

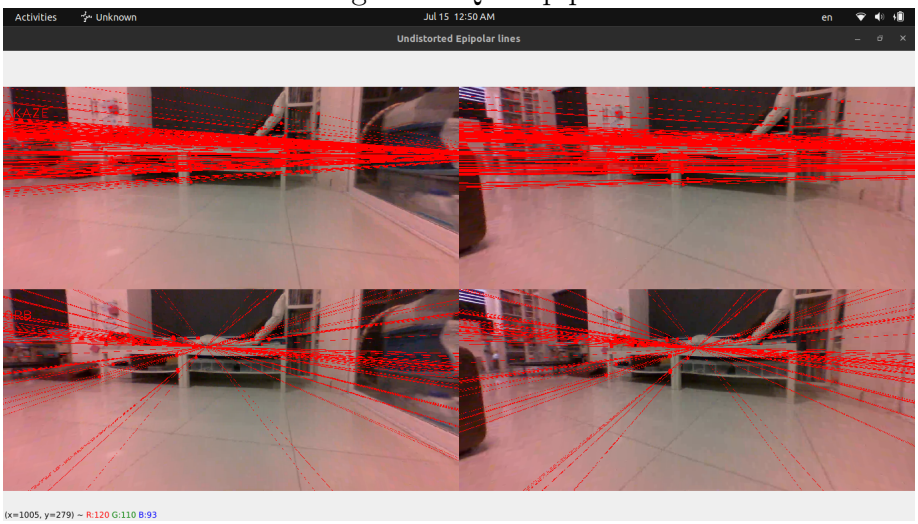



```
12      img1_new_kps = self._undistorted_key_points(
    image1_kps, img1_new_cam_mtx, img1_x, img1_y)
13
14      undistorted_img2, img2_new_cam_mtx, img2_x, img2_y
    = self._undistorted_image(image2)
15      img2_new_kps = self._undistorted_key_points(
    image2_kps, img2_new_cam_mtx, img2_x, img2_y)
16
17      res_img = cv.drawMatches(undistorted_img1,
    img1_new_kps, undistorted_img2, img2_new_kps,
    good_matches, None,
18                                      (0, 0, 255), (0, 0,
    255), flags=2)
19      return res_img
20
```

7 Q7: Epipolar Lines

```
1 def find_fundamental_matrix(self, essential_matrix):
2      mat_k = self.camera_mtx
3      mat_k_inv = np.linalg.inv(mat_k)
4      mat_k_prime = self.camera_mtx
5      mat_k_prime_inv = np.linalg.inv(mat_k_prime)
```

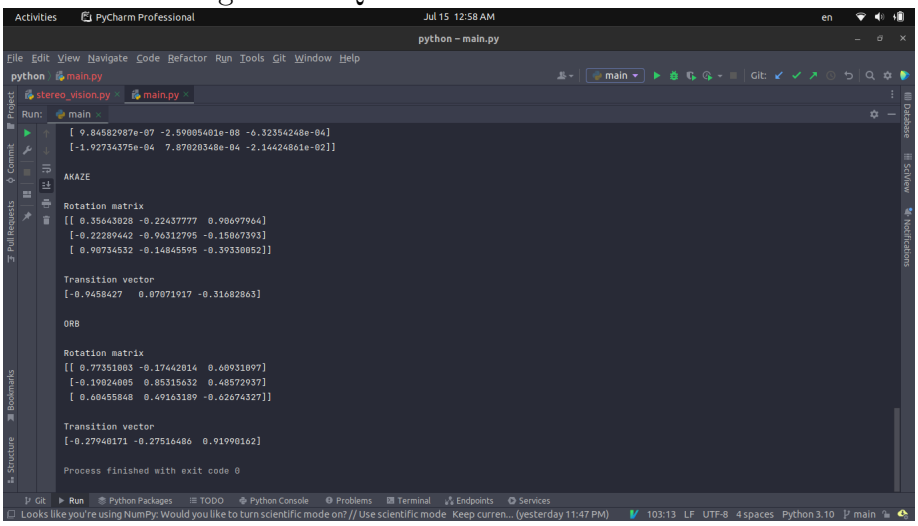
Figure 9: Q7: Epipolar Lines




```
6
7       fundamental_matrix = mat_k_inv.T @ essential_matrix
        @ mat_k_prime_inv
8       print("\nFundamental Matrix")
9       print(fundamental_matrix)
10
11      self.fundamental_matrix = fundamental_matrix
12
13      return fundamental_matrix
14
15  @staticmethod
16  def _draw_lines(image1, image2, lines, points1, points2
        , color):
17      r, c = image1.shape[:2]
18      for r, pt1, pt2 in zip(lines, points1, points2):
19          x0, y0 = map(int, [0, -r[2] / r[1]])
20          x1, y1 = map(int, [c, -(r[2] + r[0] * c) / r
        [1]])
21          image1 = cv.line(image1, (x0, y0), (x1, y1),
        color, 1)
22          image1 = cv.circle(image1, tuple(pt1), 5, color
        , -1)
23          image2 = cv.circle(image2, tuple(pt2), 5, color
        , -1)
```

```
24      return image1, image2
25
26  def find_epipolar_lines(self, image1, image2, points1,
        points2, fundamental_matrix, color):
27      img1 = image1.copy()
28      img2 = image2.copy()
29      r_epi_lines = cv.computeCorrespondEpilines(points2.
        reshape(-1, 1, 2), 2, fundamental_matrix)
30      r_epi_lines = r_epi_lines.reshape(-1, 3)
31
32      l_epi_lines = cv.computeCorrespondEpilines(points1.
        reshape(-1, 1, 2), 1, fundamental_matrix)
33      l_epi_lines = l_epi_lines.reshape(-1, 3)
34
35      pts1 = points1.astype(np.int32)
36      pts2 = points2.astype(np.int32)
37
38      image5, image6 = self._draw_lines(img1, img2,
        r_epi_lines, pts1, pts2, color)
39      image7, image8 = self._draw_lines(img2, img1,
        l_epi_lines, pts2, pts1, color)
40      res_img = cv.hconcat([image5, image7])
41
42      return res_img, l_epi_lines, r_epi_lines
43
44  def undistorted_epipolar_lines(self, image1, image2,
        image1_pts, image2_pts):
45      undistorted_img1, img1_new_cam_mtx, img1_x, img1_y
        = self._undistorted_image(image1)
46      undistorted_img2, img2_new_cam_mtx, img2_x, img2_y
        = self._undistorted_image(image2)
47
48      undistorted_pts1 = self._undistorted_points(
        image1_pts, img1_new_cam_mtx, img1_x, img1_y)
49      undistorted_pts2 = self._undistorted_points(
        image2_pts, img2_new_cam_mtx, img2_x, img2_y)
50
51      _, essential_matrix, essential_pts1, essential_pts2
        , _ = self.find_essential_matrix(undistorted_pts1,
52
                                         undistorted_pts2)
```

Figure 10: Q8: Factorization of Essential Matrix

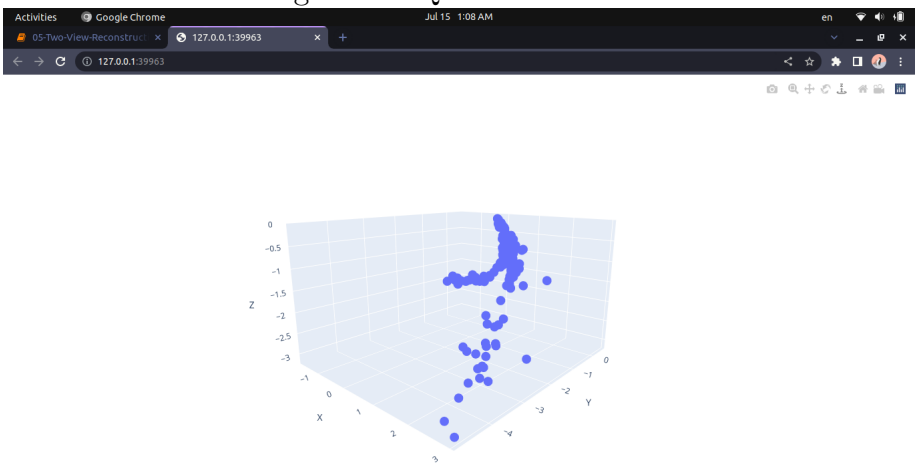



```
53      fundamental_matrix = self.find_fundamental_matrix(
        essential_matrix)
54
55      res_img, _, _ = self.find_epipolar_lines(
        undistorted_img1, undistorted_img2, undistorted_pts1
        , undistorted_pts2,
56
        fundamental_matrix, (0, 0, 255))
57      return res_img
58
```

8 Q8: Factorization of Essential Matrix

```
1 # ---------- Decompose essential matrix ----------
2 print("\nAKAZE")
3 akaze_rotation, akaze_transition = stereo_vision1.
    decompose_essential_matrix()
4 print("\nORB")
5 orb_rotation, orb_transition = stereo_vision2.
    decompose_essential_matrix()
6
7 def decompose_essential_matrix(self):
8     mat_u, mat_s, mat_vt = np.linalg.svd(self.
```

Figure 11: Q9: AKAZE Points Cloud

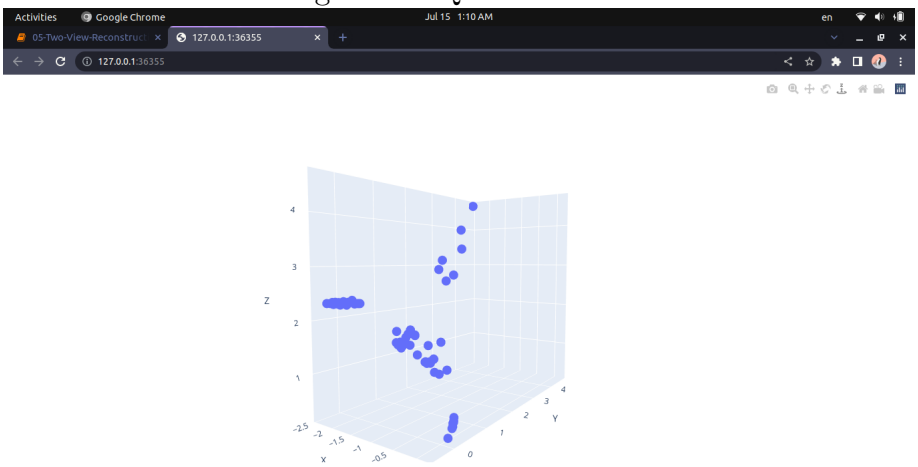



```
    essential_matrix)
 9    mat_w = np.array([
10          [0.0, -1.0, 0.0],
11          [1.0, 0.0, 0.0],
12          [0.0, 0.0, 1.0]
13    ])
14    mat_r = mat_u @ mat_w @ mat_vt
15    mat_t = mat_u[:, 2]
16
17    print("\nRotation matrix")
18    print(mat_r)
19    print("\nTransition vector")
20    print(mat_t)
21
22    self.rotation_mtx = mat_r
23    self.transition_vec = mat_t
24
25    return mat_r, mat_t
26
```

9 Q9: Recover Relative Pose

```
1 import pandas as pd
```

Figure 12: Q9: ORB Points Cloud




```
2 import plotly.express as px
3 # ------------ Recover Relative Poses ------------
4 akaze_fundamental_matrix = stereo_vision1.
      find_fundamental_matrix(akaze_essential_matrix)
5 akaze_points_3d = stereo_vision1.recover_relative_pose(
      akaze_essential_pts1, akaze_essential_pts2,
      akaze_fundamental_matrix,
6
      akaze_essential_matrix)
7
8 df = pd.DataFrame(akaze_points_3d, columns=["X", "Y", "
      Z"])
9 fig = px.scatter_3d(df, x="X", y="Y", z="Z")
10 fig.show()
11
12 orb_fundamental_matrix = stereo_vision2.
      find_fundamental_matrix(orb_essential_matrix)
13 orb_points_3d = stereo_vision2.recover_relative_pose(
      orb_essential_pts1, orb_essential_pts2,
      orb_fundamental_matrix,
14
       orb_essential_matrix)
15
16 df = pd.DataFrame(orb_points_3d, columns=["X", "Y", "Z
```

```
    "])
fig = px.scatter_3d(df, x="X", y="Y", z="Z")
fig.show()

def recover_relative_pose(self, points1, points2, fundamental_matrix, essential_matrix):
    pts1 = points1.reshape(1, -1, 2)
    pts2 = points2.reshape(1, -1, 2)

    pts1, pts2 = cv.correctMatches(fundamental_matrix, pts1, pts2)

    _, rotation, translation, mask = cv.recoverPose(essential_matrix, pts1, pts2, self.camera_mtx)
    print("\nRotation: ")
    print(rotation)
    print("\nTranslation: ")
    print(translation)

    projection1 = self.camera_mtx @ np.concatenate((rotation, translation), axis=1)
    projection2 = self.camera_mtx @ np.concatenate((np.eye(3), np.zeros((3, 1))), axis=1)
    print("\nProjection 1: ")
    print(projection1)
    print("\nProjection 2: ")
    print(projection2)

    pts1 = pts1.reshape(2, -1)
    pts2 = pts2.reshape(2, -1)

    points_4d = cv.triangulatePoints(projection1, projection2, pts1, pts2)
    points_4d = points_4d / np.tile(points_4d[-1, :], (4, 1))
    points_3d = points_4d[:3, :].T
    return points_3d

```