2023 Fall 3DCV - hw3 Report

生醫電資所 碩一 r12945040 郭思言

Briefly Explain My Method in Each Step:

Camera Calibration

I use the code provided by TA directly:

1. Read the **calib_video.avi** check board video, manually select at least 4 frames of images.
2. With `cv.findChessboardCorners()`, find the corners of the selected images.
3. Use the `cv.calibrateCamera()` function to calibrate the camera directly, and find the **Camera Intrinsic** and **Distortion Coefficient** parameters.

Feature Matching

Use ORB [Rublee 2011] as feature extractor:

- The ORB algorithm:
  - Improved FAST (features from accelerated segment test) algorithm to detect feature points.
  - Idea: if a pixel is significantly different from the neighborhood pixels then it is more likely to be a corner point.
  - Steps:
    1. Image feature point detection. (with brightness I_p and a brightness threshold T)
    2. Feature point screening: The ORB algorithm improves the original FAST algorithm which calculates the Harris response values for the original FAST corner points and sorts them according to the gray value and take the first N points.
    3. Image scale pyramids are constructed and sampled on each layer of the pyramid to extract FAST features and add scale invariance to feature

points.

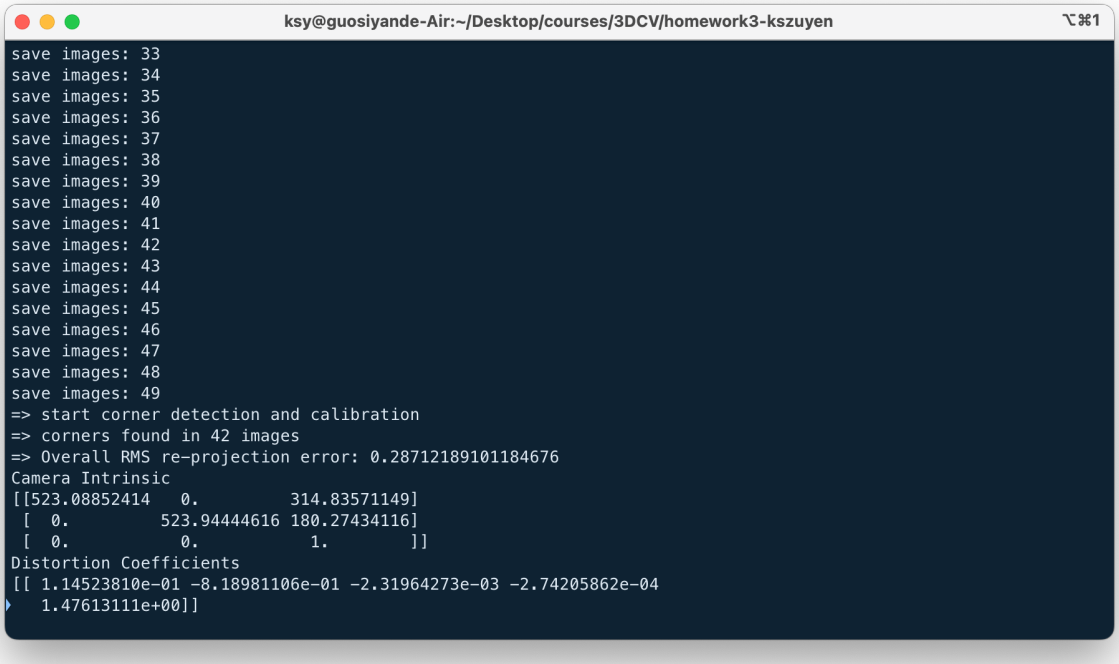
4. Determine the feature point direction, using the Intensity Centroid method. (In order to make the extracted feature points have rotational invariance)

- Implementation code: input two images, return the corresponding matching points:

```
def extract_feature(self, img1, img2):
        """extract features with orb feature extractor"""
        # Initiate ORB detector
        orb = cv.ORB_create()
        # find the keypoints and descriptors with ORB
        kp1, des1 = orb.detectAndCompute(img1, None)
        kp2, des2 = orb.detectAndCompute(img2, None)
        # create BFMatcher object
        bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)
        # Match descriptors.
        matches = bf.match(des1, des2)
        # Sort them in the order of their distance.
        matches = sorted(matches, key=lambda x: x.distance)

        points1 = np.array([kp1[m.queryIdx].pt for m in matches])
        points2 = np.array([kp2[m.trainIdx].pt for m in matches])

        return points1, points2
```



references

- https://iopscience.iop.org/article/10.1088/1742-6596/1237/3/032020/pdf
- https://docs.opencv.org/4.5.1/dc/dc3/tutorial_py_matcher.html

Pose from Epipolar Geometry (pseudo codes and comments)

I use `cv.findEssentialMat()` function to find the Essential matrix and the mask, and use `cv.recoverPose()` to find the Rotation and translation from the Essential matrix, corresponding points, camera parameters and the mask.

```
def process_frames(self, queue):
        # initialize the R, t
        prev_R, prev_t = np.eye(3, dtype=np.float64), np.zeros((3, 1), dtype=np.float64)
        # read 1st image
        img1 = self.read_image(self.frame_paths[0])

        for frame_path in self.frame_paths[1:]:
            # TODO: compute camera pose here
            img2 = self.read_image(frame_path)
            # extract features with orb feature extractor
            points1, points2 = self.extract_feature(img1, img2)
```

```
            # find essential matrix
            E, mask = cv.findEssentialMat(points1, points2, self.K, threshold=1)
            # recover pose
            retval, R, t, mask = cv.recoverPose(E, points1, points2, self.K, mask=mask)
            # calculate the relative pose
            R = R.dot(prev_R)
            t = prev_t + R.dot(t)

            queue.put((R, t))

            # Draw the matched (tracked) point on current image
            for point in points2:
                cv.circle(img2, (int(point[0]), int(point[1])), 2, (0, 255, 0), -1)

            cv.imshow("frame", img2)

            # update img1, prev_R, prev_t
            img1 = img2
            prev_R = R
            prev_t = t

            if cv.waitKey(30) == 27:
                break
```

`cv.findEssentialMat()`:

- 5 point algorithm, proposed by David Nister in 2004
  - two corresponding image points m and m'

$$\mathbf{m'^{T}Fm} = 0.$$

equation (1)

  - We know the intrinsic matrix K

$$\mathbf{K^{-T}EK^{-1}} = \mathbf{F}.$$

equation (2)

  - An essential matrix E is a faithful representation of the motion (translation and rotation, up to a scale), it has only 5 DOFs (fundamental matrix has 7). It must also satisfy 2 more constraints:

$$2\mathbf{E}\mathbf{E}^{\mathbf{T}}\mathbf{E} - tr(\mathbf{E}\mathbf{E}^{\mathbf{T}})\mathbf{E} = 0.$$

equation (3)

- Pseudocode:

```
1. Write down the epipolar equation (equation(1)) for the 5 points
2. Use the 9 equations of equation(3), form a 9 x 20 coefficient matrix corresponding to a
monomial vector:
    [x3, y3, x2y, xy2, x2z, x2, y2z, y2, xyz,
    xy, xz2, xz, x, yz2, yz, y, z3, z2, z, 1]
Then apply Gaussian-Jordan elimination to the 9 x 20 matrix, reduce it to an upper triangu
lar form.
3. Use some ad hoc procedures to extract the determinants of two 4 x 4 matrices, followed
by a second stage of elimination.
4. Back-substituting those real roots, solve the unknowns one by one to achieve E.
```

reference: http://users.cecs.anu.edu.au/~hongdong/new5pt_cameraREady_ver_1.pdf

`cv.recoverPose()`

- This function decomposes an essential matrix and verifies possible pose hypotheses by doing chirality check.
- Chirality check
  - The triangulated 3D points should have positive depth.
- Directly use this function to get R, t from essential matrix E.

Results Visualization

Youtube link: (results visualization starts from 8:11)

- https://youtu.be/qtC661V8vtc?si=UgJQRej8oOnO1Wsm

Full Demo Video:

- https://youtu.be/qtC661V8vtc

Environment

- python: version 3.9
  - numpy==1.26.1

- open3d==0.16.1
- opencv_python==4.8.1.78