HuaLiu

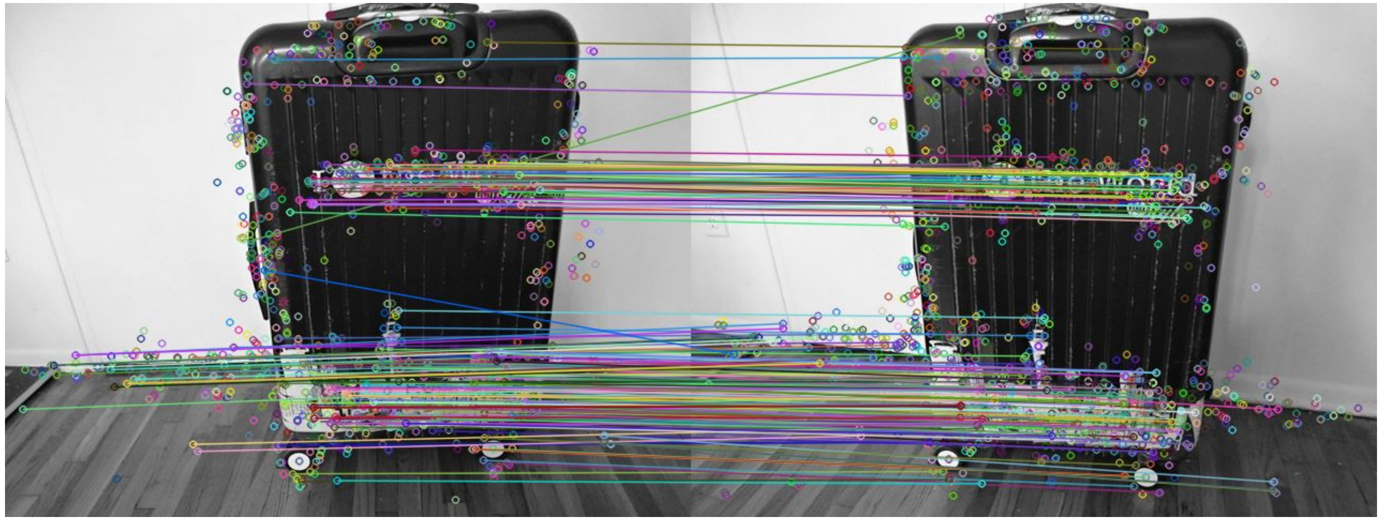
3D Reconstruction steps:

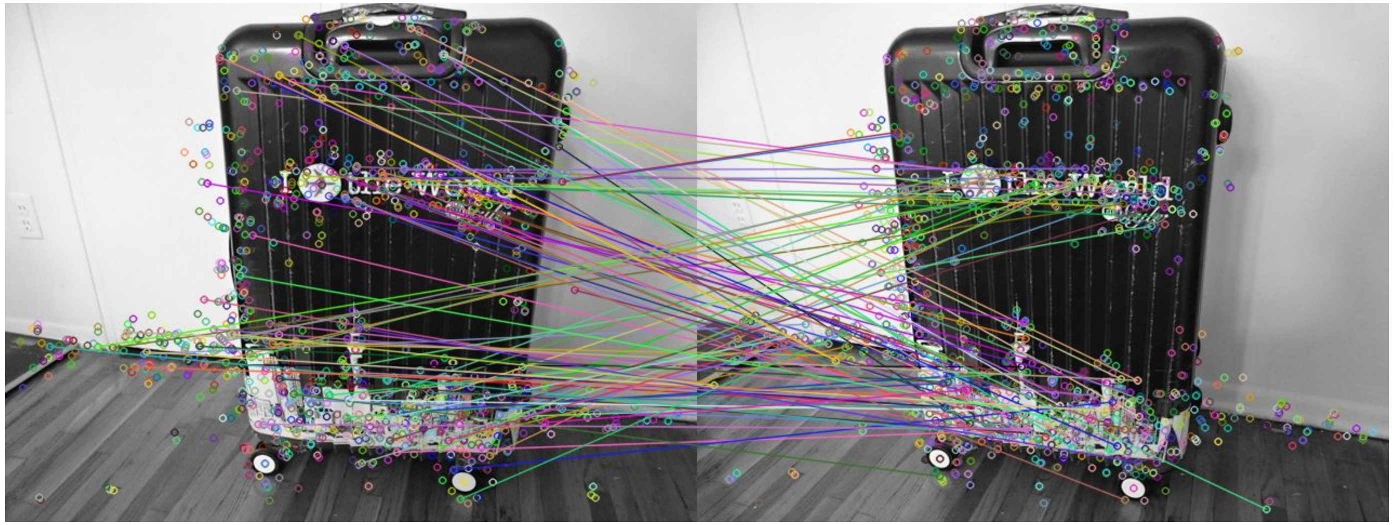
- Feature Matching
- Estimation of Camera Pose
- Triangulation between img1 and imgn
- (Sparse)3D Reconstruction from multiple views
- Augment Reality

Feature Matching ->Method one:

- 1)In all images, detect keypoints and compute descriptors using SURF
- 2)Match points between the two conjunctive images
- 3)Choose “good matches”: Calculate max and min distances between keypoints
- 4)Tracks features across all images and generise pairwiseMatches(a vector of “good matches”)
- In opencv 2)3)4) can be done with one class FeaturesMatcher:

Feature Matching ->Method one:





vector<MatchesInfo> pairwiseMatches;
matcher(features, pairwiseMatches);

using the code above, I got the result on the left.

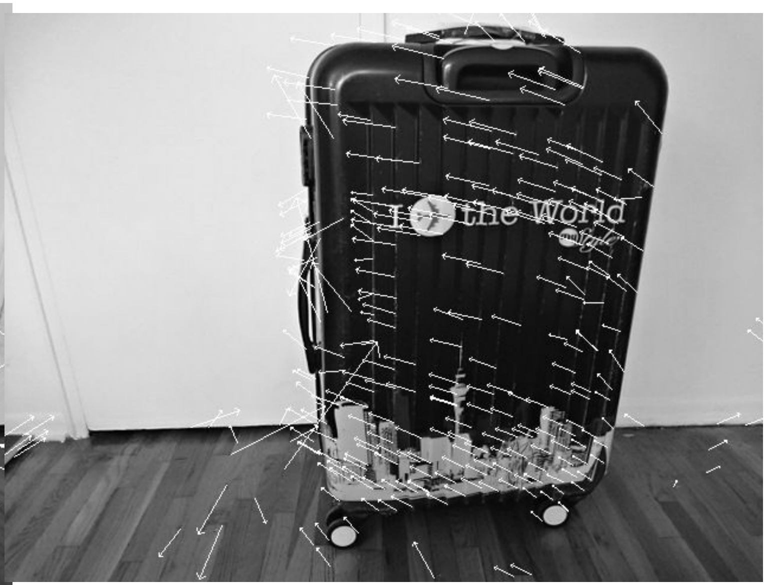
So I tried another way OpticalFlow

Feature Matching ->Method two:

- 1)In all images, detect keypoints using SURF
- 2)Convert keypoints to Point2f format and track them using OpticalFlow.

```
if (points[i + 1].size() != points[0].size())
std::cout << "Warning: Losing keypoints!!!" << endl;
```

- After test, all points size are 307.


I the World

I the World

I the World

I the World

I the World

I the World

Estimation of Camera Pose

1)Find fundamental matrix F between each image pair using RANSAC.

2)Find essential matrix E using $E = K_r^{-T} F K_l^{-1}$, where K is the optimized intrinsic matrix.

3)Assume camera matrix P1 is at origin - no rotation or translation. And Decompose E using SVD to get the second camera matrix P2 . However, by decomposing E, you can only get the direction of the translation. So there are generally 4 possible solutions(HZ 9.19). of which we select the one that results in reconstructed 3D points in front of both cameras.

Estimation of Camera Pose

- This step can be done by cv::decomposeEssentialMat(E, R1, R2, t);and Generally 4 possible seconde camera poses exists P2:[R1,t], [R1,−t], [R2,t], [R2,−t]. Then get the final one with positive depth after triangulation.
- Alternatively, it can be done by recoverPose(E, points[0], points[1], R, t, focalLength, principalPoint);This command decomposes E and get the final R,t using Cheirality check, which means that the triangulated 3D points should have positive depth.
- I chose the second one.

Absolute pose

- tFst = tFst + (RFst.t())*t;
- RFst = RFst*R;//1R3=1R2*2R3


```
t: !!opencv-matrix
  rows: 3
  cols: 1
  dt: d
  data: [ 6.7953757063460074e-001, 1.1809354653687600e+000,
    1.8567888153598671e+000 ]
RFst: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [ -2.4945316593717393e-001, -6.2589906381595972e-001,
    7.3893401594338959e-001, -6.4225092769300485e-001,
    6.7802227330562448e-001, 3.5749061914817837e-001,
    -7.2476676516062433e-001, -3.8540389050420570e-001,
    -5.7111905702999133e-001 ]
```

Triangulation between img1 and imgn

1)cv::undistortPoints(points[i], points[i], cameraMatrix, distortion);

2)construct projection matrix(intrinsic * extrinsic) P2 = cameraMatrix*P2;

3)cv::triangulatePoints(P1, P2, points[i], points[i+1], points4D);

(Sparse)3D Reconstruction from multiple views

1)caculate all imagepairs and take the average

2)Since this is based on the assumption that the first camera matrix is[I|t]. So there is a scale factor. solvePnP.

Augment Reality

- 1) find chess board in image
- 2)generate 3d points, with z=0
- 3)Use solvePnP to get extrinsic matrix
  rvec, tvec

cv::solvePnP(obj, corners, cameraMatrix, distortion, rvec, tvec);

4)cv::projectPoints(obj2, rvec, tvec, cameraMatrix, distortion, corners2);