Image Features & Matching

Sung Soo Hwang

Image matching

- Result

Image matching

- Example code

```
int main(){
	Mat query, image, descriptors1, descriptors2;
	Ptr<ORB> orbF = ORB::create(1000);
	vector<KeyPoint> keypoints1, keypoints2;
	vector< vector< DMatch> > matches; //DMatch is descriptor match
	vector< DMatch > goodMatches;
	BFMatcher matcher(NORM_HAMMING);
	Mat imgMatches;

	int i, k;
	float nndr;

	query = imread("assets/query.jpg");
	image = imread("assets/input.jpg");
	resize(query, query, Size(640, 480));
	resize(image, image, Size(640, 480));

	if (query.empty() || image.empty()) return -1;
```

Image matching

- Example code

```
//Compute ORB Features
orbF->detectAndCompute(query, noArray(), keypoints1, descriptors1);
orbF->detectAndCompute(image, noArray(), keypoints2, descriptors2);

//KNN Matching(k-nearest neighbor matching)
//Find best and second-best matches
k = 2;
matcher.knnMatch(descriptors1, descriptors2, matches, k);

// Find out the best match is definitely better than the second-best match
nndr = 0.6f;
for (i = 0; i < matches.size(); i++) {
        if (matches.at(i).size() == 2 && matches.at(i).at(0).distance <= nndr * matches.at(i).at(1).distance) {
                goodMatches.push_back(matches[i][0]);
        }
}
```

Image matching

- (Side Note)

```
// Find out the best match is definitely better than the second-best match
nndr = 0.6f;
for(i = 0; i < matches.size(); i++) {
        if (matches.at(i).size() == 2 && matches.at(i).at(0).distance <= nndr * matches.at(i).at(1).distance) {
                goodMatches.push_back(matches[i][0]);
        }
}
```

Because k is 2

The best match's distance in i^{th} matches

2^{nd} best match's distance in i^{th} matches

If best match's distance is much smaller than 2^{nd} best match's distance, store best value.
Else, throw away best and 2^{nd} best. (Because they are too close so we cannot determine good match)

Nearest Neighbor Distance Ratio should be close to 0 to be a "good match". (1 == worst)
If it is above 0.6, it is not a good match.

NNDR == Best / 2^{nd} best

Best / 2^{nd} best < 0.6 ➔ Good match

= Best < 0.6 * 2^{nd} best ➔ Good match

Image matching

- Example code

```
        //Draws the found matches of keypoints from two images.
        drawMatches(query, keypoints1, image, keypoints2, goodMatches, imgMatches,
        Scalar::all(-1), Scalar(-1), vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);

        if (goodMatches.size() < 4) { cout << "Matching failed" << endl; return 0; }

        imshow("imgMatches", imgMatches);
        waitKey(0);
}
```

Image matching

- drawMatches()

```
void cv::drawMatches ( InputArray                img1,
                       const std::vector< KeyPoint > & keypoints1,
                       InputArray                img2,
                       const std::vector< KeyPoint > & keypoints2,
                       const std::vector< DMatch > & matches1to2,
                       InputOutputArray          outImg,
                       const Scalar &            matchColor = Scalar::all(-1) ,
                       const Scalar &            singlePointColor = Scalar::all(-1) ,
                       const std::vector< char > & matchesMask = std::vector< char >() ,
                       int                       flags = DrawMatchesFlags::DEFAULT
                     )
```

- **img1**: First source image.
- **keypoints1**: Keypoints from the first source image.
- **img2**: Second source image.
- **keypoints2**: Keypoints from the second source image.

Image matching

- drawMatches()
  - **matches1to2**: Matches from the first image to the second one, which means that keypoints1[i] has a corresponding point in keypoints2[matches[i]] .
  - **outImg**: Output image. Its content depends on the flags value defining what is drawn in the output image. See possible flags bit values below.
  - **matchColor**: Color of matches (lines and connected keypoints). If matchColor==Scalar::all(-1) , the color is generated randomly.
  - **singlePointColor**: Color of single keypoints (circles), which means that keypoints do not have the matches. If singlePointColor==Scalar::all(-1) , the color is generated randomly.
  - **matchesMask**: Mask determining which matches are drawn. If the mask is empty, all matches are drawn.
  - **flags**: Flags setting drawing features. Possible flags bit values are defined by DrawMatchesFlags.