Homework#10

2015004302 곽상원

목차

1. Purpose of program
2. Experimental process
3. Result
4. Discussion

---

1. Purpose of program

- 동일한 환경을 프레임을 달리해 얻은 2개의 이미지에 대해 **Nonlinear data fitting**을 실시한다.
- Data set을 구하는 방식은, 각 이미지에서 key points를 얻어낸다. 각 이미지의 key points에서 비슷한 성향을 지니는 점을 match points로 삼는다. 그 결과로 data set of (x,y,x',y')를 얻을 수 있다.
- Nonlinear fitting model은 다음과 같다.

Model: 2D transformation between given images

$$x' = \frac{a_{11}x + a_{12}y + a_{13}}{a_{31}x + a_{32}y + 1}$$

$$y' = \frac{a_{21}x + a_{22}y + a_{23}}{a_{31}x + a_{32}y + 1}$$

이때, Levenberg-Marquardt method를 이용해 set of a = (a11,a12,a13,a21,a22,a23,a31,a32)를 구한다.(이 후, 아래에서는 표기의 편의를 위해 set of a=(a1,a2,a3,a4,a5,a6,a7,a8)로 정의한다.)

▶Levenberg-Marquardt method

1. initial guess로 a_cur를 정한다.
2. a_cur에 대한 error square를 구한다.
3. 적당한 r(ex) r=0.001)을 정하고,

$$H' \triangle a = -\nabla x^2(a_{cur}) \text{ (이 때, } H' = H + rI, \nabla x^2(a_{cur}) \text{는 } gredient)$$

를 이용해 a_next를 정한다.

4. 만약 a_next에 대한 error square가 a_cur에 대한 error square보다 크거나 같다면 r값을 10배하고 3으로 돌아간다. 아닌 경우, a를 updata하고 r값을 1/10배하고 3으로 돌아간다.

2 .Experimental process

source file을 2개의 프로젝트로 나눠서, 첫 번째 프로젝트(opencv_test_source)는 images에서 matching points를 찾아내고 이 결과물을 text file에 출력한다. 두 번째 프로젝트(data_fitting_source)는 text file을 읽어 set of a를 찾는다.

< Nonlinear datafitting >

goodLeft.jpg, goodRight.jpg에 대해,

① Noise가 없는 경우 ② Gaussian noise(SD=1.0) ③ Gaussian noise(SD=10.0)

④ Gaussian noise(SD=30.0)

⑤ **badLeft.jpg, badRight.jpg**에 대해 Noise가 없는 경우를 포함해 총 5개의 cases에 대해 set of a를 구한다.

▶Levenberg-Marquardt 적용

1. intial guess set of a=(1.0,1.0, ... ,1.0)T, r=0.001로 set

```
//initial guess a= (1.0, ...)T & initial r set
        for (int i = 0; i < 9; i++) {
                a[i] = 1.0;
                k[i] = 0.0;
        }
        r = 0.001;
```

2. Gradient 및 Hessian matrix구하기

```
//gradient 구하기
        for (int i = 0; i < N; i++) {
                float C, D, E;
                x = Left[i].x, y = Left[i].y, x_ = Right[i].x, y_ = Right[i].y;
                C = (a[1] * x + a[2] * y + a[3]) / (a[7] * x + a[8] * y + 1);
                D = (a[4] * x + a[5] * y + a[6]) / (a[7] * x + a[8] * y + 1);
                E = 1.0 / (a[7] * x + a[8] * y + 1.0);
                k[1] += -2.0 * (x_ - C) * x * E; k[2] += -2.0 * (x_ - C) * y * E; k[3] += -2.0 *
(x_ - C) * 1.0 * E;
                k[4] += -2.0 * (y_ - D) * x * E; k[5] += -2.0 * (y_ - D) * y * E; k[6] += -2.0 *
(y_ - D) * 1.0 * E;
                k[7] += 2.0 * (x_ - C) * x * E * C + 2.0 * (y_ - D) * x * E * D;
                k[8] += 2.0 * (x_ - C) * y * E * C + 2.0 * (y_ - D) * y * E * D;
        }
//-1곱하기
        for (int i = 1; i <= 8; i++) k[i] *= -1.0;
```

```
//Hessian 구하기
	for (int i = 0; i < N; i++) {
		float E, temp1[8], temp2[8];//temp1은 x'편미분, temp2는 y'편미분 모아놓음
		x = Left[i].x, y = Left[i].y, x_ = Right[i].x, y_ = Right[i].y;
		E = 1.0f / (a[7] * x + a[8] * y + 1.0f);
		temp1[0] = x * E, temp1[1] = y * E, temp1[2] = E, temp1[3] = temp1[4] = temp1[5] = 0.0;
		temp1[6] = (a[1] * x + a[2] * y + a[3]) * (-x) * E * E, temp1[7] = (a[1] * x + a[2] * y + a[3]) * (-y) * E * E;
		temp2[0] = temp2[1] = temp2[2] = 0.0, temp2[3] = x * E, temp2[4] = y * E, temp2[5] = E;
		temp2[6] = (a[4] * x + a[5] * y + a[6]) * (-x) * E * E, temp2[7] = (a[4] * x + a[5] * y + a[6]) * (-y) * E * E;
		for (int e = 0; e < 8; e++) for (int h = 0; h < 8; h++) H[e + 1][h + 1] = temp1[e] * temp1[h] + temp2[e] * temp2[h];
		}
//H'구하기
	for (int i = 1; i <= 8; i++) H[i][i] += r;
```

3. error square 비교를 통해 a와r값 update

```
//a에 의해 error가 어떻게 변화했는지 구하고, 상황에 따른 처리
	float before = 0.0, after = 0.0;
	for (int i = 0; i < N; i++) {
		x = Left[i].x, y = Left[i].y, x_ = Right[i].x, y_ = Right[i].y;
		before += ((x_ - (a[1] * x + a[2] * x + a[3]) / (a[7] * x + a[8] * y + 1))) * ((x_ - (a[1] * x + a[2] * x + a[3]) / (a[7] * x + a[8] * y + 1)))
			+ ((y_ - (a[4] * x + a[5] * x + a[6]) / (a[7] * x + a[8] * y + 1))) * ((y_ - (a[4] * x + a[5] * x + a[6]) / (a[7] * x + a[8] * y + 1)));
		after += ((x_ - ((a[1] + k[1]) * x + (a[2] + k[2]) * x + (a[3] + k[3])) / ((a[7] + k[7]) * x + (a[8] + k[8]) * y + 1))) * ((x_ - ((a[1] + k[1]) * x + (a[2] + k[2]) * x + (a[3] + k[3])) / ((a[7] + k[7]) * x + (a[8] + k[8]) * y + 1)))
			+ ((y_ - ((a[4] + k[4]) * x + (a[5] + k[5]) * x + (a[6] + k[6])) / ((a[7] + k[7]) * x + (a[8] + k[8]) * y + 1))) * ((y_ - ((a[4] + k[4]) * x + (a[5] + k[5]) * x + (a[6] + k[6])) / ((a[7] + k[7]) * x + (a[8] + k[8]) * y + 1)));
	}
	if (before <= after) r *= 10.0;
	else {
		for (int i = 1; i <= 8; i++) a[i] += k[i];
		r /= 10.0;
	}
```

4. relative approximate error가 10^-4보다 작을 경우 종료

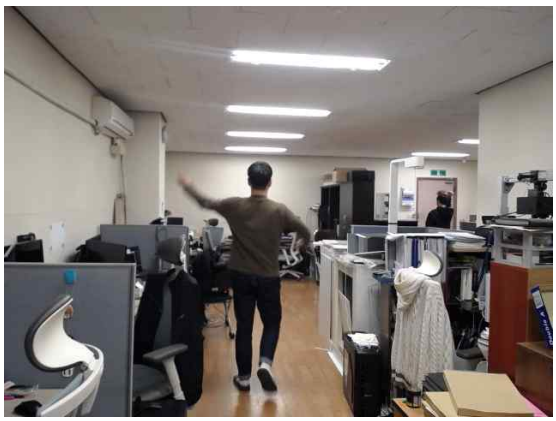
```
//더 이상 a에 변화가 없을 시에 종료
                if (100 * (after - before) / after < 10E-4) {
                        printf("Algorithm end!!\n");
                        break;
                }
```

3.Result

▶Images는 다음과 같다.

< goodLeft&Right >


< badLeft&Right >




▶각 cases에 대한 set of a는 다음과 같다.

goodLeft.jpg, goodRight.jpg에 대해,

① Noise가 없는 경우(test_no_noise.txt)

```
Algorithm end!!
------test_no_noise.txt------
Matching points: 180
Set of a:
a[1]:   1.256
a[2]:   1.191
a[3]:   1.001
a[4]:   1.282
a[5]:   1.247
a[6]:   1.002
a[7]:   0.463
a[8]:   0.562
Error square: 7122991.00000
```

② Gaussian noise(SD=1.0)(test_sd_1.txt)

```
Algorithm end!!
------test_sd_1.txt------
Matching points: 184
Set of a:
a[1]:   1.263
a[2]:   1.196
a[3]:   1.001
a[4]:   1.289
a[5]:   1.252
a[6]:   1.002
a[7]:   0.448
a[8]:   0.552
Error square: 7318515.50000
```

③ Gaussian noise(SD=10.0)(test_sd_10.txt)

```
Algorithm end!!
------test_sd_10.txt------
Matching points: 176
Set of a:
a[1]:   1.245
a[2]:   1.186
a[3]:   1.001
a[4]:   1.275
a[5]:   1.245
a[6]:   1.002
a[7]:   0.480
a[8]:   0.569
Error square: 6915584.50000
```

④ Gaussian noise(SD=30.0)(test_sd_30.txt)

```
Algorithm end!!
------test_sd_30.txt------
Matching points: 152
Set of a:
a[1]:    1.223
a[2]:    1.167
a[3]:    1.001
a[4]:    1.242
a[5]:    1.208
a[6]:    1.001
a[7]:    0.535
a[8]:    0.625
Error square: 6215356.00000
```

badLeft.jpg, badRight.jpg에 대해

⑤ Noise가 없는 경우(test_bad.txt)

```
Algorithm end!!
------test_bad.txt------
Matching points: 67
Set of a:
a[1]:    1.130
a[2]:    1.082
a[3]:    1.001
a[4]:    1.105
a[5]:    1.071
a[6]:    1.001
a[7]:    0.765
a[8]:    0.847
Error square: 2947040.00000
```

4.Discussion

- 각 cases의 accuracy를 알아보기 위해 set of a를 이용해 error square를 구했는데, 이때 각 case의 matching points의 개수가 차이가 나서 정확한 비교는 어렵지만 (Error square)/(Matching points)를 이용했다.

case	(Error square)/(Matching points)
1	39572.17
2	39774.54
3	39293.09
4	40890.50
5	43985.67

기존의 image에 분산이 큰 noise가 추가될수록 더 큰 error가 발생함을 확인할 수 있었다.

또한 bad image가 good image에 비해 평균적으로 더 큰 error가 발생한다.

그 이유로는 noise가 첨가되거나 이미지에 변형이 많을수록 Matching points중에 data fitting curve를 크게 벗어나는 outlier가 증가하기 때문이라 추측된다.