Computer vision - Problem set 3

Nicolas Six

March 1, 2018

Question 1 Calibration

1.1 Computing M from ground truth

```python
def gen_a(zipped_point):
    n = len(zipped_point)

    a = np.zeros(shape=(2 * n, 12), dtype=np.double)

    for i, ((p_x, p_y), (w_x, w_y, w_z)) in
        enumerate(zipped_point):
        a[2 * i, :] = [w_x, w_y, w_z, 1, 0, 0, 0, 0,
            -p_x * w_x, -p_x * w_y, -p_x * w_z, -p_x]
        a[2 * i + 1, :] = [0, 0, 0, 0, w_x, w_y, w_z, 1,
            -p_y * w_x, -p_y * w_y, -p_y * w_z, -p_y]

    return a


def compute_projection_matrix(zipped_point):
    a = gen_a(zipped_point)
    ata = np.matmul(a.T, a)
    val, vec = np.linalg.eig(ata)
    i = np.argmin(val)
    sol = vec[:, i]
    factor = 1 / np.sum(np.abs(np.power(sol, 2)))
    sol *= factor
    return np.reshape(sol, newshape=(3, 4))
```

$$M = \begin{bmatrix} -0.45827554 & 0.29474237 & 0.01395746 & -0.0040258 \\ 0.05085589 & 0.0545847 & 0.54105993 & 0.05237592 \\ -0.10900958 & -0.17834548 & 0.04426782 & -0.5968205 \end{bmatrix}$$

$$\begin{bmatrix} u & v \end{bmatrix} = \begin{bmatrix} 0.14190608 & -0.45184301 \end{bmatrix}$$

1.2 Compute M from randomly sampled point and ground truth

```python
def compute_m_from_sample(full_point_list):
    min_diff = None
    best = None
    diff_list = []
    for s in [8, 12, 16]:
        for i in range(10):
            random.shuffle(full_point_list)
            m =
                compute_projection_matrix(full_point_list[:s])
            diff = 0.0
            for n in range(1, 5):
                p = full_point_list[-n]
                p3d = np.ones(shape=4)
                p3d[0:3] = p[1]
                p2d = np.ones(shape=3)
                p2d[0:2] = p[0]
                conv = np.matmul(m, p3d)
                conv /= conv[2]
                diff += np.sum(np.power(p2d - conv, 2))
            if min_diff is None or min_diff > diff:
                min_diff = diff
                best = (m, s, i)
            diff_list.append(diff / 4)
    return best, diff_list
```

As you can see on Table 1, the more point you take to build the estimator the lower is the maximum error. But at the same time minimum value as a tendency to rise, as you include more and more outliers to build our model. Therefor, most of the run we executed select a matrix from the $k = 12$ column. This explain why the RANDSAC algorithm try a lot of combination of the smallest possible subset.

$$M = \begin{bmatrix} 6.91763681e-03 & -3.99581895e-03 & -1.36628106e-03 & -8.27786171e-01 \\ 1.54257321e-03 & 1.02297155e-03 & -7.25995610e-03 & -5.60924953e-01 \\ 7.58555207e-06 & 3.71409261e-06 & -1.93437070e-06 & -3.38125837e-03 \end{bmatrix}$$

1.3 Camera world coordinate

```python
c = - np.matmul(np.linalg.inv(m[0:3, 0:3]), m[:, 3])
```

$$C = \begin{bmatrix} 303.10580556 & 307.17684589 & 30.42320286 \end{bmatrix}$$

Table 1: Residual value get for ten run

Run	$k = 8$	$k = 12$	$k = 16$
1	3.081	1.873	2.090
2	8.669	4.972	2.399
3	3.331	2.340	0.949
4	3.936	2.360	1.408
5	1.803	1.183	1.003
6	21.197	1.978	1.848
7	1.978	5.080	2.762
8	3.036	0.727	2.081
9	0.858	0.597	4.547
10	5.527	1.971	0.542

Question 2 Fundamental Matrix Estimation

2.1 Least square estimation of fundamental matrix

```
def gen_a(zipped_point):
    n = len(zipped_point)

    a = np.zeros(shape=(n, 9), dtype=np.double)

    for i, ((a_x, a_y), (b_x, b_y)) in
        enumerate(zipped_point):
         a[i, :] = [a_x * b_x, a_x * b_y, a_x,
                    a_y * b_x, a_y * b_y, a_y,
                    b_x,       b_y,       1]
    return a


def compute_projection_matrix(zipped_point):
    a = gen_a(zipped_point)
    ata = np.matmul(a.T, a)
    val, vec = np.linalg.eig(ata)
    i = np.argmin(val)
    sol = vec[:, i]
    factor = 1 / np.sum(np.abs(np.power(sol, 2)))
    sol *= factor
    return np.reshape(sol, newshape=(3, 3))
```

$$\tilde{F} = \begin{bmatrix} -6.60698417e-07 & 7.91031620e-06 & -1.88600198e-03 \\ 8.82396296e-06 & 1.21382933e-06 & 1.72332901e-02 \\ -9.07382303e-04 & -2.64234650e-02 & 9.99500092e-01 \end{bmatrix}$$

2.2 Rank reduced version of fundamental matrix

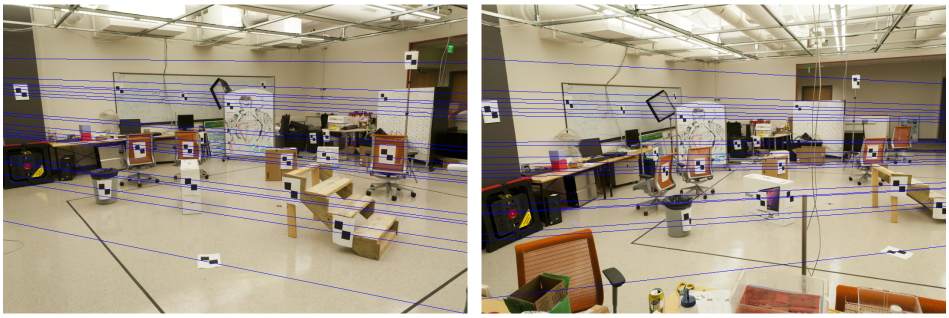
```
def reduce_rank(f):
    u, d, v = np.linalg.svd(f)
    d[2] = 0.0
    return np.mat(u) * np.mat(np.diag(d)) * np.mat(v)
```

$$F = \begin{bmatrix} -5.36264198e-07 & 7.90364770e-06 & -1.88600204e-03 \\ 8.83539184e-06 & 1.21321685e-06 & 1.72332901e-02 \\ -9.07382265e-04 & -2.64234650e-02 & 9.99500092e-01 \end{bmatrix}$$

2.3 Drawing epipolar lines

```
def point_of_line(p, f):
    ph = np.mat([0, 0, 1])
    ph[0, :2] = p[:2]
    fx = np.mat(f) * ph.T
    b = 0
    p1 = ((b * fx[1] + fx[2]) / -fx[0], b)
    b = 712
    p2 = ((b * fx[1] + fx[2]) / -fx[0], b)
    return p1, p2


def draw_lines(img, src_pts, f):
    for pt in src_pts:
        p1, p2 = point_of_line(pt, f)
        p1 = tuple(map(int, p1))
        p2 = tuple(map(int, p2))
        cv2.line(img, p1, p2, (255, 0, 0), 1)
    return img
```



(a) Image A

(b) Image B

Figure 1: Epipolar line computed from the point list provided, drawn on the original image