MATHEMATICAL FOUNDATIONS OF COMPUTER GRAPHICS AND VISION

Exercise 1. Robust Estimation and Optimization

Dongho Kang

March 12, 2017

The version of MATLAB which was used and tested is as follows:
The "code" directory contains followings:

- ***part1.m*** Script .m file for part1.
- ***part2.m*** Script .m file for part2.
- ***functions dir*** Function .m files
  - ...
  - ...

For running a exercise part, adjust several parameters and run corresponding script on the MATLAB environment. More details are stated in the *Running* section of each exercise part.

1 EXERCISE PART 1: RANSAC FOR CIRCLE FITTING

1.1 DESCRIPTION

In exercise part 1, RANSAC algorithm was used for circular model fitting with N number of 2D data points which were corrupted by noise and certain ration of outliers. The implementation of part 1 corresponds to the .m script file ***part1.m*** and the script consists of several parts which can be executed separately:

- Data generation and verification
- Running RANSAC algorithm for circular model fitting

- Running exhaustive search for circular model fitting
- Plotting the result

1.1.1 Data generation and verification

For certain outlier ratio r(%) and N number of 2D data points, the number of inliers (n_{inlier})and outliers($n_{outlier}$) are as follows:

$$n_{inlier} = N \times \frac{1-r}{100}$$
$$n_{outlier} = N \times \frac{r}{100}$$
$$(N = 100,\ r \in \{5, 20, 30, 70\})$$

In order to generate n_{inlier} inliers and $n_{outlier}$ outliers, the function **GenerateInlierData** and **GenerateOutlierData** are implemented. **GenerateInlierData** generates noise-added data points which is not over the inlier distance threshold $\tau = 0.1$ from synthetic model and **Generate-OutlierData** generates data points which is over τ. It can be described as follows. $d\big(p_1, p_2\big)$ is euclidean distance function between two 2D points $p_1 and p_2$:

$$\Big|d\big((x_i, y_i), (x_c, y_c)\big) - R_c\Big| \le \tau \qquad for (x_i, y_i) \in Set_{inlier}$$
$$\Big|d\big((x_j, y_j), (x_c, y_c)\big) - R_c\Big| > \tau \qquad for (x_i, y_i) \in Set_{outlier}$$

After generating inliers/outliers, verified whether generated inliers is threshold and also generated outliers . For this purpose, the function **VerifyInlier** is used. The specific descriptions for each functions are following:

Inlier generation A 2D point $(\tilde{x}_i, \tilde{y}_i)$ on the circular model $(x - x_c)^2 + (y - y_c)^2 = R_c^2$ which of center (x_c, y_c), and radius R_c can be described as follows:

$$(\tilde{x}_i, \tilde{y}_i) = (x_c, y_c) + R_c \times (\cos\theta_i, \sin\theta_i) \qquad \theta_i \in [0, 2\pi], i \in \{1, 2, \dots n_{inlier}\}$$

Thus, picked n_{inlier} number of random float numbers from the range of $[0, 2\pi]$ by MATLAB function **rand** and as using these numbers as θ_i, generated (x_i, y_i) for $i \in \{1, 2, \dots n_{inlier}\}$. Random noise $(\sigma_{x_i}, \sigma_{y_i})$ is added on the data point $(\tilde{x}_i, \tilde{y}_i)$ as follows:

$$(x_i, y_i) = (\tilde{x}_i, \tilde{y}_i) + (\sigma_{x_i}, \sigma_{y_i}) \quad s.t. \quad \sigma_{x_i}, \sigma_{y_i} \in [-0.1, 0.1], \quad \Big|d\big((x_i, y_i), (x_c, y_c)\big) - R_c\Big| \le \tau$$

To implement this, two random float numbers by **rand** were picked for $(\sigma_{x_i}, \sigma_{y_i})$ iteratively, until it satisfies $\Big|d\big((x_i, y_i), (x_c, y_c)\big) - R_c\Big| \le \tau$ and repeated it for $i = 1 ... n_{inlier}$ to generate n_{inlier} number of inliers. Also, (x_i, y_i) is checked whether it is in the domain $[-10, 10]$.

OUTLIER GENERATION In order to generate one outlier point (x_j, y_j), two random float numbers were generated by **rand** in the domain of $[-10,10] \times [-10,10]$ and generated (x_j, y_j) was checked whether it satisfies $\left|d((x_j, y_j), (x_c, y_c)) - R_c\right| > \tau$. If not, generated (x_j, y_j) again. Repeated this for $n_{outlier}$ times to generate $n_{outlier}$ number of outliers.

DATA VERIFICATION Checked whether synthesized inliers are indeed inliers and outliers are indeed outliers. The function **VerifyInlier** was used. The function calculates $|d((\tilde{x}_i, \tilde{y}_i), (x_c, y_c)) - R_c|$ for each data points and check if there is a inlier point which of $|d((x_i, y_i), (x_c, y_c)) - R_c| > \tau$ or an outlier point which of $\left|d((x_j, y_j), (x_c, y_c)) - R_c\right| \leq \tau$. In this case, error is returned and the script halts.

1.1.2 RANSAC FOR CIRCULAR MODEL FITTING

For generated data point (x_i, y_i) $i \in 1, 2 \ldots, N$, RANSAC algorithm applied to get the best model parameter which of maximum number of inliers within n_{iter} RANSAC iterations. **RansacForCircularModel** was implemented for RANSAC algorithm and the algorithm is applied for 1000 times for each outlier ratio to find the distribution of the number of inliers found by RANSAC.

The number of RANSAC iterations n_{iter} for success rate p ($0 \leq p < 1$, for this exercise, $p = 0.99$), outlier ratio $r(\%)$ and sample size s (= 3 for circular model: two for x, y coordinates of center and one for radius) is defined as follows:

$$n_{iter} = \lceil \frac{log(1-p)}{log(1-(1-\frac{r}{100})^s)} \rceil$$

Thus, $n_{iter} = 3$ for $r = 5\%$, $n_{iter} = 7$ for $r = 20\%$, $n_{iter} = 11$ for $r = 30\%$ and $n_{iter} = 169$ for $r = 70\%$. During n_{iter} iterations, found best result of RANSAC which of maximum number of inliers.

For each RANSAC iteration, sampled $s = 3$ data points without replacement. If the same data points are sampled, model fitting could be failed because it is under-determined problem.

CIRCULAR MODEL FITTING With respect to sampled data points $(x'_1, y'_1), (x'_2, y'_2), (x'_3, y'_3)$, used **CircularModelFitting** for model fitting. The function solves the following equation (variables a, b, c are parameter for fitted model which correspond to x_c, y_c and R_c each):

$$\begin{aligned}(x'_1 - a)^2 + (y'_1 - b)^2 = c^2 \\ (x'_2 - a)^2 + (y'_2 - b)^2 = c^2 \\ (x'_3 - a)^2 + (y'_3 - b)^2 = c^2\end{aligned}$$

As eliminating variable c, the equation can be expressed to linear equation as follows:

$$\begin{bmatrix} -2(x_1' - x_2') & -2(y_1' - y_2') \\ -2(x_2' - x_3') & -2(y_2' - y_3') \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x_2'^2 - x_1'^2 + y_2'^2 - y_1^2 \\ x_3'^2 - x_2'^2 + y_3'^2 - y_2^2 \end{bmatrix}$$

For $Ax = b$, if A is square matrix and not singular $x = A^{-1}b$. Thus $x = [a \quad b]^T$ can be obtained and c also be calculated by the following equation:

$$c = \sqrt{x_1^2 - 2ax_1 + a^2 + y_1^2 - 2by_1 + b^2}$$

As a, b and c (center and radius of the fitted circle) is determined, set of inliers and outliers can be determined as being done for data verification. This circular model fitting algorithm can be used for not only RANSAC but also for the exhaustive search.

1.1.3 Exhaustive search

Another option for model fitting is trying every combination of data points and choosing the best combination. In order to do this, MATLAB function **nchoosek** was used. For $N = 100$ and $s = 3$, the number of all possible combination is as follows:

$$n_{comb} = \binom{100}{3} = \frac{100!}{3! \times 97!} = 161700$$

For each combination, used **CircularModelFitting** for model fitting and finding the number of inliers for the fitted model. Finally, as compared result of each combinations, the best result of exhaustive search can be obtained. This was implemented as the function **ExhSearchForCircularModel**.

1.2 Running

Several parameters can be changed for running the script but only changing model parameters (i.e. variables x_c, y_c, radius, noise_radius) is allowed to guarantee proper execution. If model parameter combination is not feasible(e.g. out of domain bound), error would be returned.

1.3 Result

image

1.4 Discussion

The answers of the questions regarding RANSAC and exhaustive search are as follows:

- The number of combinations for $N = 100$

$$n_{comb} = \binom{100}{3} = \frac{100!}{3! \times 97!} = 161700$$

- The number of RANSAC iterations for each outlier ratio for $N = 100$

$$n_{iter} = \lceil \frac{log(1-p)}{log(1-(1-\frac{r}{100})^s)} \rceil$$

$r =$	5%	20%	30%	70%
$n_{iter} =$	3	7	11	169

- The number of combinations for $N = 100,000$

$$n_{comb} = \binom{100000}{3} = \frac{100000!}{3! \times 99997!} = 166661666700000$$

- The number of RANSAC iterations for $N = 100,000$ is same as the case of $N = 100$
- TODO

2 Exercise part 2: IRLS and norms for line fitting

2.1 Description

In exercise part 2, IRLS and Linear Programming algorithm was used for line model fitting with N number of 2D data points which were corrupted by noise and certain ration of outliers. The implementation of part 2 corresponds to the .m script file ***part2.m*** and the script consists of several parts:

- Data generation and verification
- Run IRLS with L_1 norm
- Run LP with L_1 norm
- Run LP with L_∞ norm
- Plot the result

2.1.1 Data generation and verification

To generate data points on a line, the new cost function(vertical cost: **Cost**) was used instead of euclidean distance function. For data point (x_i, y_i) and line model $y = ax + b$, it is defined as follows:

$$d\big((x_i, y_i), (a, b)\big) = \big|y_i - (ax_i + b)\big|$$

Thus, inliers and outliers are defined as follows:

$$d\big((x_i,y_i),(a,b)\big) \le \tau \qquad for(x_i,y_i) \in Set_{inlier}$$
$$d\big((x_j,y_j),(a,b)\big) > \tau \qquad for(x_i,y_i) \in Set_{outlier}$$

The strategy of data generation is almost same as exercise part 1 but line model and the new cost are adapted.

Inlier generation A 2D point $(\tilde{x_i},\tilde{y_i})$ on the line model $y = ax + b$ can be described as follows:

$$\tilde{y_i} = a\tilde{x_i} + b \qquad \tilde{x_i} \in [-10,10]$$

Random noise $(\sigma_{x_i},\sigma_{y_i})$ is added on the data point $(\tilde{x_i},\tilde{y_i})$ as follows:

$$(x_i,y_i) = (\tilde{x_i},\tilde{y_i}) + (\sigma_{x_i},\sigma_{y_i}) \quad s.t. \quad \sigma_{x_i},\sigma_{y_i} \in [-0.1,0.1], \quad d\big((x_i,y_i),(a,b)\big) \le \tau$$

Outlier generation An outlier point (x_j,y_j) is generated by two random float numbers and should be in the domain of $[-10,10]\times[-10,10]$ Also it is checked whether it satisfies $d\big((x_j,y_j),(x_c,y_c)\big) > \tau$.

Data verification Inliers and outliers are verified whether satisfies the following conditions:

$$d\big((x_i,y_i),(a,b)\big) \le \tau \qquad for(x_i,y_i) \in Set_{inlier}$$
$$d\big((x_j,y_j),(a,b)\big) > \tau \qquad for(x_i,y_i) \in Set_{outlier}$$

2.1.2 IRLS with L_1 norm

For model parameter $\mathbf{x} = (a,b)^T$ and data $\mathbf{y}_i = (x_i,y_i)^T$ for $i = 1,2\ldots\ N$, the line fitting problem can be expressed with L_1 norm:

$$\operatorname*{argmin}_{\mathbf{x}} C_\rho(\mathbf{x})$$
$$where \quad C_\rho(\mathbf{x}) = \sum_{i=1}^{N} \rho \circ f_i(\mathbf{x}) = \sum_{i=1}^{N} d(\mathbf{y}_i,\mathbf{x})$$

The form above can be changed to weighted least squares problem as defining $f_i(\mathbf{x}) = d(\mathbf{y}_i,\mathbf{x})^2$ and $\rho(s) = \sqrt{s}$ and solved by IRLS(Iteratively Reweighted Least Squares):

$$\mathbf{x}^{t+1} = \operatorname*{argmin}_{\mathbf{x}} C(\mathbf{x},\mathbf{w}^t) = \operatorname*{argmin}_{\mathbf{x}} \sum_{i=1}^{N} w_i^t f_i(\mathbf{x})$$
$$where \quad w_i^t = \frac{1}{2} d(\mathbf{y}_i,\mathbf{x}^t)^{-1}$$

In the function **IRLSWithL1Norm**, the problem was described as linear equation:

$$\mathbf{x}^{t+1} = \arg\min_{\mathbf{x}} C(\mathbf{x}, \mathbf{w^t}) = \arg\min_{\mathbf{x}} \sum_{i=1}^{n} w_i^t f_i(\mathbf{x}) = \arg\min_{\mathbf{x}} (A\mathbf{x} - b)^T W^t (A\mathbf{x} - b) = (A^T W^t A)^{-1} A^T W^t b$$

$$where$$

$$W^t = diag(w_1^t, w_2^t, \ldots, w_N^t), \quad A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix}, \quad b = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

As iterating until $\left\| \mathbf{x}^{t+1} - \mathbf{x}^t \right\| < tol$ (tol is an error tolerance), $\mathbf{x} = (a, b)^T$ can be obtained.

2.1.3 LP WITH L_1 NORM

The line fitting problem with L_1 norm:

$$\arg\min_{\mathbf{x}} \sum_{i=1}^{N} d(\mathbf{y}_i, \mathbf{x}) = \arg\min_{\mathbf{x}} \left\| A\mathbf{x} - b \right\|_1 = \arg\min_{\mathbf{x}} \sum_{i=1}^{N} \left| A_i \mathbf{x} - b_i \right|$$

$$where$$

$$A_i = \begin{bmatrix} x_i & 1 \end{bmatrix}, \quad b_i = \begin{bmatrix} y_i \end{bmatrix}$$

It can be solved by linear programming algorithm:

$$\min_{\mathbf{x}, t_1, \ldots, t_N} \sum_{i=1}^{N} t_i$$

$$s.t. \quad \left| A_i \mathbf{x} - b_i \right| \le t_i, \quad i = 1, \ldots, N$$

The MATLAB function **linprog(A, b, f)** can be used:

$$\mathbf{A} = \left[\begin{array}{cc|c} -x_1 & -1 & \\ -x_2 & -1 & \\ \vdots & \vdots & -\mathbb{1}_{N\times N} \\ -x_N & -1 & \\ \hline x_1 & 1 & \\ x_2 & 1 & \\ \vdots & \vdots & -\mathbb{1}_{N\times N} \\ x_N & 1 & \end{array}\right], \quad \mathbf{b} = \left[\begin{array}{c} -y_1 \\ -y_2 \\ \vdots \\ -y_N \\ \hline y_1 \\ y_2 \\ \vdots \\ y_N \end{array}\right], \quad \mathbf{f} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \quad \mathbf{x'} = \begin{bmatrix} a \\ b \\ t_1 \\ \vdots \\ t_N \end{bmatrix}$$

As **linprog(A, b, f)** solves $\min_{\mathbf{x'}} \mathbf{f}^T \mathbf{x'}$ such that $\mathbf{A}\,\mathbf{x} \le \mathbf{b}$, the line parameter $\mathbf{x} = (a, b)^T$ can be obtained from **x'**.

2.1.4 LP WITH L_∞ NORM

The line fitting problem with L_∞ norm:

$$\arg\min_{\mathbf{x}} \|A\mathbf{x} - b\|_\infty = \arg\min_{\mathbf{x}} \max_{i=1,\dots,N} |A_i\mathbf{x} - b_i|$$
$$where$$
$$A_i = \begin{bmatrix} x_i & 1 \end{bmatrix}, \quad b_i = \begin{bmatrix} y_i \end{bmatrix}$$

It can be solved by linear programming algorithm:

$$\min_{\mathbf{x},t} t$$
$$s.t. \quad |A_i\mathbf{x} - b_i| \le t, \quad i = 1,\dots,N$$

Thus, **A**, **B**, **f** and **x'** for **linprog(A, b, f)** are as follows:

$$\mathbf{A} = \left[\begin{array}{ccc} -x_1 & -1 & -1 \\ -x_2 & -1 & -1 \\ \vdots & \vdots & \vdots \\ -x_N & -1 & -1 \\ \hline x_1 & 1 & -1 \\ x_2 & 1 & -1 \\ \vdots & \vdots & \vdots \\ x_N & 1 & -1 \end{array}\right], \quad \mathbf{b} = \left[\begin{array}{c} -y_1 \\ -y_2 \\ \vdots \\ -y_N \\ \hline y_1 \\ y_2 \\ \vdots \\ y_N \end{array}\right], \quad \mathbf{f} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{x'} = \begin{bmatrix} a \\ b \\ t \end{bmatrix}$$

The line parameter $\mathbf{x} = (a, b)^T$ can be obtained from **x'**

2.2 RUNNING

2.3 RESULT

image

2.4 DISCUSSION