ETH Zürich
Mathematical Foundations of
Computer Graphics and Vision
Spring 2019

M. Arda Düzceker
18-950-485
ardad@student.ethz.ch

EXERCISE 1

ROBUST ESTIMATION AND OPTIMIZATION

1. RANSAC FOR CIRCLE FITTING

In this part of the exercise, a RANSAC algorithm and an exhaustive search algorithm are implemented to fit a circle to various generated data which consists of different outlier ratios.

To run this section, *../code/part1/* must be the working directory. Then, running *part1.m* performs all the necessary calculations required, outputs the results to command window and generates the plot.

1.1. Data Generation

Generation of the underlying circle model for inliers is randomized. *generateRandomCircleModel* function returns the center coordinates and the radius of the circle model which the data is sampled from later. The domain for the data is restricted to be in [−10, 10] × [−10, 10], therefore for visual purposes, the center coordinates of the circle model is restricted to be in [−5, 5] × [−5, 5] and the range of the randomized radius is adjusted so that the circle is inside the domain and the radius is larger than 3 (clearly visible). These parameters can be modified inside the function.

Generation of the data utilizing the model is also randomized. *generateCircleData* takes the model parameters (x_c, y_c, R), desired outlier ratio (r), inlier-outlier distance threshold ($\tau = 0.1$) and returns the data samples. For this purpose, polar coordinates are used. A point on the circle can be defined as:

$$(x, y) = (R * cos\theta + x_c, R * sin\theta + y_c)$$

For inliers, required number of θ values are randomly sampled from the range $[0, 2\pi]$ and required number of noise values are randomly sampled from the range [-0.1, 0.1]. The noise is added to the R values of each data point to ensure that after the addition of noise, an inlier point's distance to the circle is below $\tau = 0.1$. Consequently, coordinates of an inlier data sample can be defined as:

$$x_i = (R \pm 0.1) * cos\theta_i + x_c$$

$$y_i = (R \pm 0.1) * sin\theta_i + y_c$$

For outliers, the coordinates of the points are randomly sampled from [−10, 10] × [−10, 10] domain. However, to ensure the correct number of outliers, a point's distance to the circle is calculated as:

$$d = \left|\sqrt{(x_i - x_c)^2 + (y_i - y_c)^2} - R\right|$$

If $d \leq 0.1$, the point is discarded and regenerated. The process continues until desired number of outliers are reached.

M. Arda Düzceker
18-950-485
ardad@student.ethz.ch

1.2. RANSAC

To perform one RANSAC run on a given data, *ransacCircle* function is called. The function takes the data matrix, number of iterations and the inlier-outlier threshold as parameters and returns the parameters of the fitted circle that results in maximum number of inliers at any iteration.

For circle fitting minimum sample set size $s = 3$ meaning that given three points one can find only one circle that goes through all three of them unless they lie on the same line (if so, R = ∞). Also, confidence (p) is given as 0.99 in the exercise. So, the number of iterations (N) that *ransacCircle* takes as a parameter for different values of outlier ratio (r) is calculated as:

$$N = \left\lceil \frac{\log(1-p)}{\log(1-(1-r)^s)} \right\rceil$$

Therefore, for outlier ratios 0.05, 0.2. 0.3, 0.7, the number of iterations required to achieve at least 99% confidence is 3, 7, 11, 169 respectively.

At each iteration of a RANSAC run, three points are sampled randomly and *fitCircle* function is called in which using the circle equation $a(x^2 + y^2) + bx + cy + 1 = 0$, three equations are written in homogenous form $Az = 0$:

$$Az = \begin{bmatrix} x_1^2 + y_1^2 & x_1 & y_1 & 1 \\ x_2^2 + y_2^2 & x_2 & y_2 & 1 \\ x_3^2 + y_3^2 & x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ 1 \end{bmatrix} = 0$$

Right null-space of A gives the circle coefficients. Then,

$$x_{c_{fit}} = -\frac{b}{2a}$$

$$y_{c_{fit}} = -\frac{c}{2a}$$

$$R_{fit} = \frac{\sqrt{b^2 + c^2 - 4a}}{2a}$$

In each iteration, the number of inliers for the fitted circle is found. Finally, the best fit $(x_{c_{fit}}, y_{c_{fit}}, R_{fit})$ giving the maximum number of inliers is returned as well as the coordinates of the inlier points.

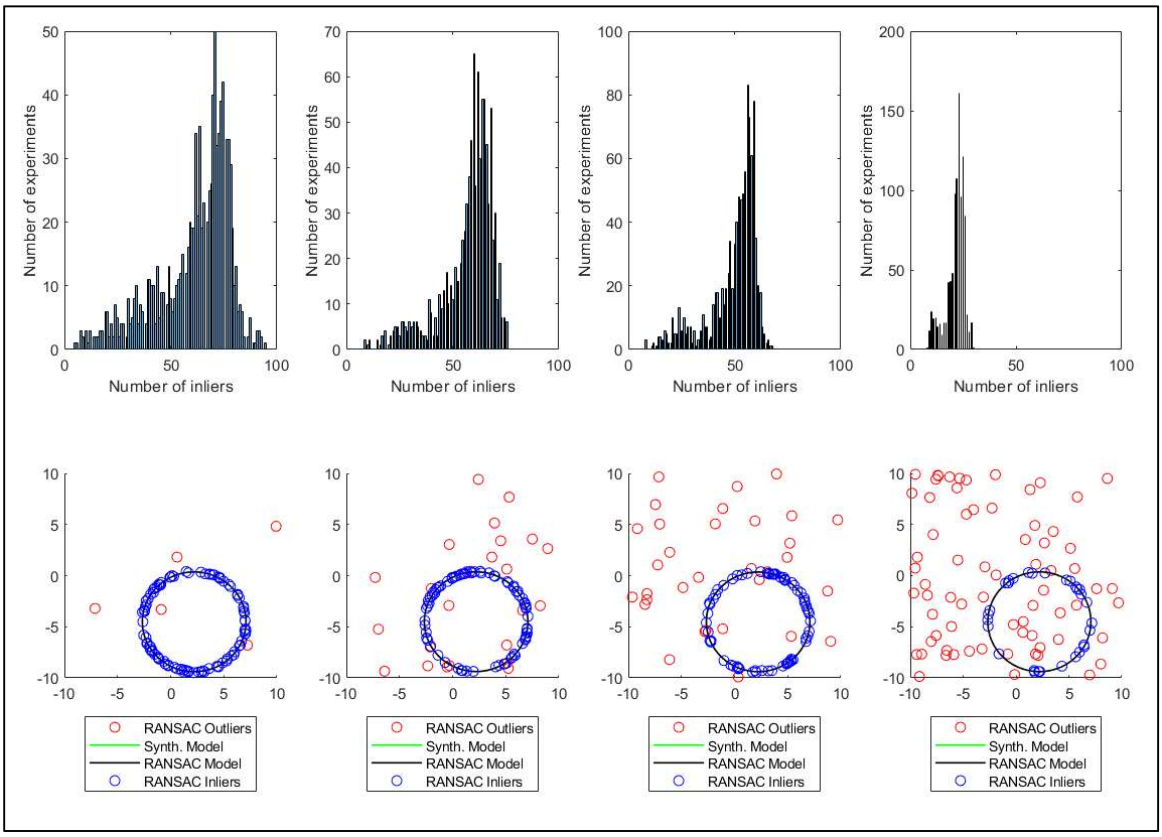
1.3. Exhaustive Search

Exhaustive search is performed inside *part1.m* code itself. The only difference from RANSAC procedure is that instead of sampling three random points for a number of iterations, every possible minimal set is attempted. Therefore, the number of iterations (N) is determined by the number of points (n) and minimal set size (s). In our case:

$$N = \binom{n}{s} = \binom{100}{3} = 161700$$

M. Arda Düzceker
18-950-485
ardad@student.ethz.ch

1.4. Results and Discussion



	TRUE MODEL	5% OUTLIER		20% OUTLIER	
		RANSAC	EXHAUSTIVE	RANSAC	EXHAUSTIVE
Circle Center:	(2.209 , -4.515)	(2.215 , -4.516)	(2.215 , -4.516)	(2.215 , -4.500)	(2.213 , -4.509)
Circle Radius:	4.876	4.880	4.880	4.873	4.877
Number of Found Inliers:	--	95	95	76	78
Time Elapsed (s):	--	0.113	2.751	0.138	2.876

	TRUE MODEL	30% OUTLIER		70% OUTLIER	
		RANSAC	EXHAUSTIVE	RANSAC	EXHAUSTIVE
Circle Center:	(2.209 , -4.515)	(2.218 , -4.522)	(2.210 , -4.521)	(2.207 , -4.507)	(2.208 , -4.498)
Circle Radius:	4.876	4.872	4.880	4.876	4.872
Number of Found Inliers:	--	68	68	30	30
Time Elapsed (s):	--	0.189	2.721	2.662	2.720

The time elapsed values shown for RANSAC runs are the total time spent for 1000 tests. Therefore, one RANSAC run only takes about 1/1000 of the shown value.

- For $N = 100$: exhaustive search performs 161,700 iterations.

  For $N = 100{,}000$: exhaustive search would perform 166,661,666,700,000 iterations which is impractical. Since exhaustive search attempts for all combinations, the number of iterations does not depend on outlier ratio.

  However, number iterations for RANSAC does not depend on the size of the dataset. Instead, it only depend on the parameters in the equation that is discussed above.

- Since the number of iterations are significantly smaller for RANSAC (3, 7, 11, 169 respectively), it is much faster than exhaustive search.
- Exhaustive search finds the global optimum for the strategy of fitting a circle to only three points and the best fit that RANSAC can achieve is what exhaustive search surely finds (as if confidence for RANSAC $p = 1.0$). Whereas, RANSAC may return a local optimum so, it may not be correct to trust RANSAC results after only one run as the histograms suggest. Statistically, the distribution of the number of inliers shows that one RANSAC run may end up with a quite wrong fit although the confidence is 0.99. One should either perform multiple independent runs as we do or increase the confidence depending on the data.

2. IRLS and NORMS FOR LINE FITTING

In this part of the exercise, a RANSAC algorithm and an exhaustive search algorithm are implemented to fit a circle to various generated data which consists of different outlier ratios.

To run this section, *../code/part2/* must be the working directory. Then, running *part2.m* performs all the necessary calculations required and generates the plot. Different from first part of the exercise, the code is divided into sections and can be run independently. So, if one wants to keep the model same and change the outlier ratio, then s/he run only the respective sections.

2.1. Data Generation

Generation of the underlying line model ($y = ax + b$) is randomized like the first part of the exercise. *generateRandomLineModel* function returns the a, b values of the model which the data is sampled from later. To be able to produce any line equation (except x = c), the polar representation (or Hough transformed) of a line is utilized [1].

For inliers, x values are sampled randomly from [-10, 10] range and yhe noise is added to the y values of each data point. Consequently, y coordinate of an inlier data sample can be defined as:

$$y_i = a * x_i + b \pm 0.1$$

For outliers, the coordinates of the points are randomly sampled from $[-10, 10] \times [-10, 10]$ domain. However, to ensure the correct number of outliers, a point's vertical distance to the line is calculated and if $d \leq 0.1$, the point is discarded and regenerated. The process continues until desired number of outliers are reached.

2.2. IRLS and LP Implementations

For the implementation of IRLS with L_1 norm, the lecture slides on robust optimization[2] and Wikipedia[3,4] is used. As the overview of the equation that are used where $\beta = \begin{bmatrix} a \\ b \end{bmatrix}$ and the model $y = \beta X$:

[1] https://docs.opencv.org/2.4.13.7/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html
[2] https://graphics.ethz.ch/teaching/mathfound19/slides/01_RobustOpt.pdf
[3] https://en.wikipedia.org/wiki/Iteratively_reweighted_least_squares
[4] https://en.wikipedia.org/wiki/Weighted_least_squares

M. Arda Düzceker
18-950-485
ardad@student.ethz.ch

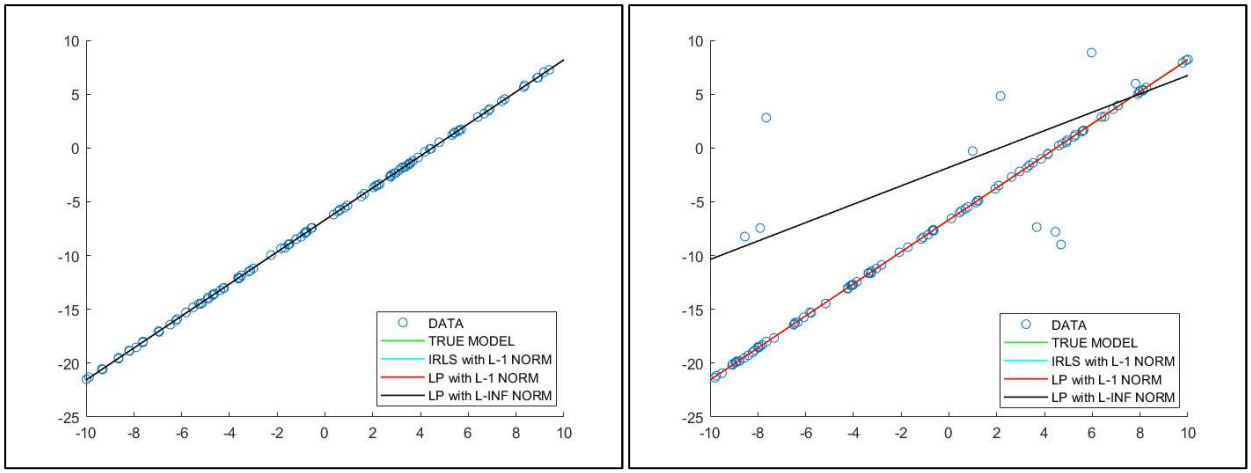
$$w_i = \frac{1}{2} * d(x, y_i)^{-1} = \frac{1}{2} * \frac{1}{|ax_i + b - y_i|}$$

$$W_{ij} = w_i, i = j \qquad W_{ij} = 0, i \neq j$$

$$\beta^{(t+1)} = \left(X^T W^{(t)} X\right)^{-1} X^T W^{(t)} y$$

For the implementation of linear programming with L_1 (*linearProgL1.m*) norm and L_∞ (*linearProgLinf.m*) norm [5] and [6] are used both of which explains the expression of the optimization problem well.

2.3. Results and Discussion



	TRUE MODEL (y = ax + b)	0% OUTLIER			10% OUTLIER		
		IRLS with L-1	LP with L-1	LP with L-INF	IRLS with L-1	LP with L-1	LP with L-INF
a:	1.4484	1.4882	1.4882	1.4881	1.4894	1.4894	0.8552
b:	-6.6884	-6.6953	-6.6953	-6.6866	-6.6728	-6.6728	-1.8124
Cost:	--	4.8124	4.8124	0.0978	91.7551	91.7551	11.1689

- Since the tolerance $\left\|\beta^{(t+1)} - \beta^{(t)}\right\|$ of my IRLS implementation is set to be same with the *linprog* implementation of the MATLAB, IRLS with L_1 and LP with L_1 generally give the same results. Without outliers, all three algorithms provide a good fit.
- With 10% outliers, we can observe that the fittings using L_1 norm result in a robust result, while L_∞ norm gets dramatically affected by outliers.
- Using L_∞ norm minimization means not allowing any error to be very large. Such a system strongly penalizes extreme outliers causing cost to be large around true model. Therefore, an optimal solution where $\max(|ax_i + b - y_i|)$ minimized is found at the black line on the left figure.

[5] http://cs.brown.edu/people/pfelzens/engn2520/CS1420_Lecture_4.pdf

[6] https://math.stackexchange.com/questions/2589887/how-can-the-infinity-norm-minimization-problem-be-rewritten-as-a-linear-program