



Understanding RANSAC

1.1 Ransac for line fitting

Choosing the parameters

- Initial number of points s
 - Typically minimum number needed to fit the model
- Distance threshold t
 - Choose t so probability for inlier is p (e.g. 0.95)
 - Zero-mean Gaussian noise with std. dev. σ : $t^2 = 3.84\sigma^2$
- Number of samples N
 - Choose N so that, with probability p , at least one random sample is free from outliers (e.g. $p=0.99$) (outlier ratio: e)

(1) What value will you choose for the initial number of points s and why?

=> According to the above picture, the initial number of points is defined as the minimum points needed to fit the model. That is, for line fitting, I chose to set $s=2$ since **two points are the minimum number to create a line**.

(2) What would be an appropriate choice for the threshold t ?

=> After doing some experiments, I found out that setting the threshold to 0.25~0.32 (I eventually chose 0.27) will cover 60%~70% of the data points. I considered that this threshold setting is reasonable because outliers above 40% are like saying almost half of the data points are outliers.

(3) What should be the value of the number of samples N so that with probability $p = 0.9999$, at least one sample is free from outliers?

=> $N = \log(1 - p) / \log(1 - (1 - e)^s) = \log(1 - p) / \log(1 - (\text{inlier} / \text{total dataCnt})^s)$

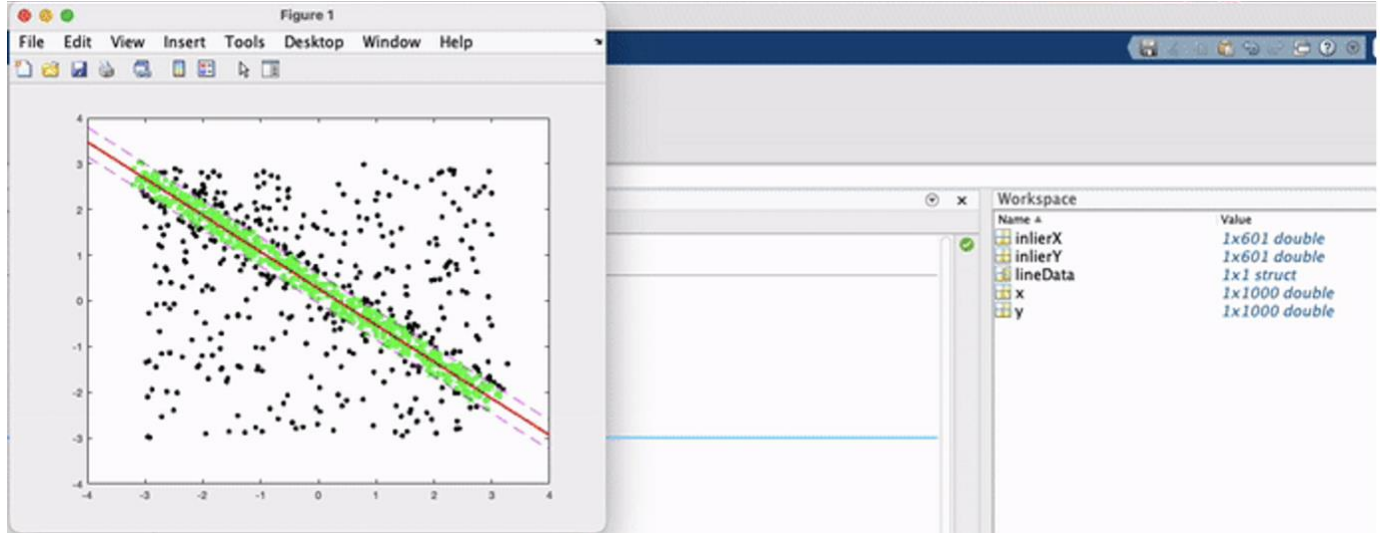
N when threshold sets at 0.25 and $p=0.9999 \Rightarrow \log(0.0001) / \log(1 - (0.601)^2) = 20.5512$

N when threshold sets at 0.27 and $p=0.9999 \Rightarrow \log(0.0001) / \log(1 - (0.611)^2) = 19.7088$

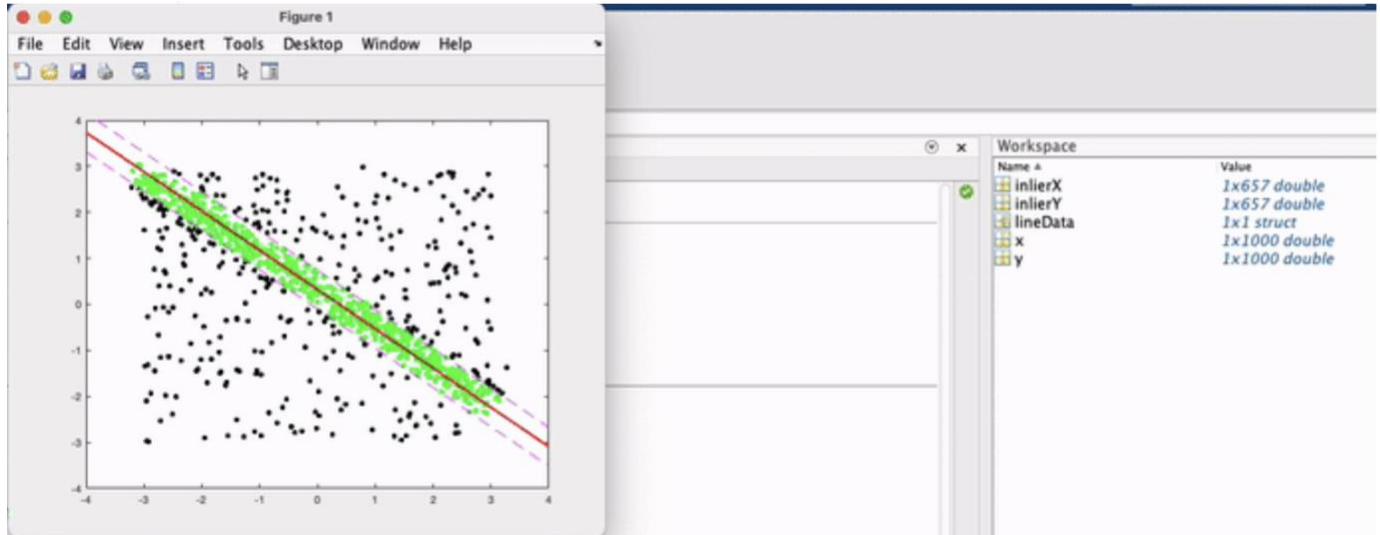
N when threshold sets at 0.32 and $p=0.9999 \Rightarrow \log(0.0001) / \log(1 - (0.657)^2) = 16.301$

(4) Result

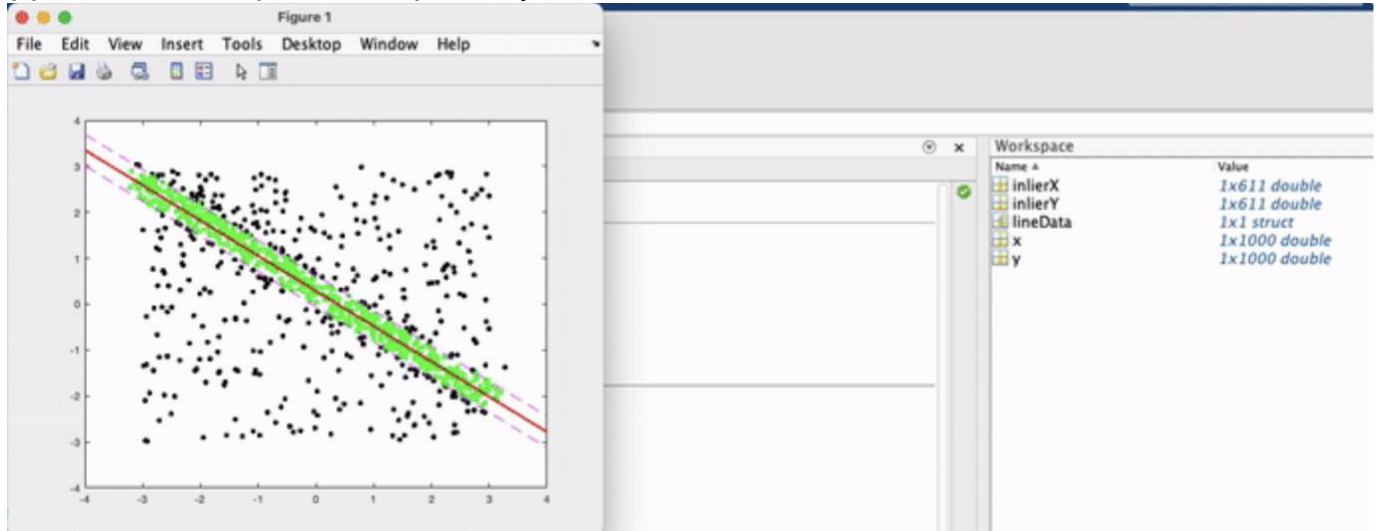
(a) Threshold 0.25 => Output 60.1% Inliers



(b) Threshold 0.32 => Output 65.7% inliers



(c) Threshold 0.27 (final choice) => Output 61.1% inliers



(5) Issue

Threshold setting is the biggest issue for me to in this question. I try to figure out the threshold in multiple ways, such as looking at the standard deviation of x and y or searching on the Internet to see if there is any rule of thumbs on threshold setting. **I found out that the threshold will be a value that is near the standard deviation of the data points (the threshold value in this question is 1.77) but lower.** I eventually decide the threshold value by seeing the ratio of the output number of the inlier points to determine what threshold value is reasonable.

(6) Surprise

I think RANSAC is a decent method to find out the potential inliers among all data points since it can be super robust and provide similar results every time I iterate.

1.2 Ransac for Affine Fitting

(1) What value will you choose for the initial number of points s and why?

=> I set the initial number of points to 2 (**s=2**) since we are trying to **fetch two points (one from the original image and the other from the transformed image) then form a line from it.**

(2) What would be an appropriate choice for the threshold t?

=> After doing some experiments, I found out that setting the threshold to **230~320 (eventually pick 260)** will make RANSAC more stable to produce an image that after applying RANSAC and `imtransform()` looks mostly similar to the transformed image.

(3) What should be the value of the number of samples N so that with probability $p = 0.9999$, at least one sample is free from outliers?

=> $N = \log(1 - p) / \log(1 - (1 - e)^s) = \log(1 - p) / \log(1 - (\text{inlier} / \text{total dataCnt2})^s)$

N when threshold sets at 260 and $p=0.9999 \Rightarrow \log(0.0001) / \log(1 - (0.7394)^2) = 11.6406$

(4) Result => Threshold 260, Output 73.94%



Compare the results with different Thresholds
Threshold = 60

Result After RANSAC



Transform Image



Threshold = 160

Result After RANSAC



Transform Image



Threshold = 260 (Best)

Result After RANSAC



Transform Image



Threshold = 360

Result After RANSAC



Transform Image



(5) Implementation

First, I will use RANSAC to find out the inliers of the original image and transformation image. Then, I will randomly pick three points and calculate the affinity matrix x ($Ax = B$). Lastly, using `imtransform()` to show the resulting image and compared it with the transformed image.

(6) Issue

At first, I do not know how to combine RANSAC with the question. Not until I reconsidered RANSAC is a function to detect outliers did I realize that the question is asking us to gather all of the

inliers of the two images after performing Spatially Invariant Feature Transform (SIFT), and utilizing the inliers to reconstruct the image. Moreover, **determining the value of the threshold is also an issue for me in this problem.** Eventually, I refer to the standard deviation of the input data (≈ 303) and find out that setting the threshold from 230~320 will give us the best results.

(7) Surprise

I think the result of this question is cool since we could find out a systematic way to filter out the outliers and match all the inlier points by using the concept of linear algebra to reconstruct the image.

Distance Transform and Chamfer Matching

2.1 Distance Transform

(1) Compute the Canny edge map image of the cow and display the result.

=> After turning "cow.png" to a grayscale image and implementing Canny edge detection on the image



(2) Compute the L1 distance transform image of the cow with your code and display the result.

=> After performing L1 distance transform and not yet converting the result to binary form









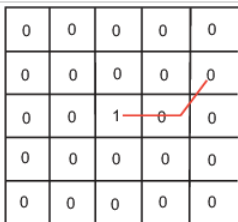

=> After converting the result into binary form with `mat2gray()`



- (3) Compute the distance transform of the cow edge map with three different metrics using `bwdist`. Comment on the differences between these. Under what circumstances would the various distance metrics be useful?

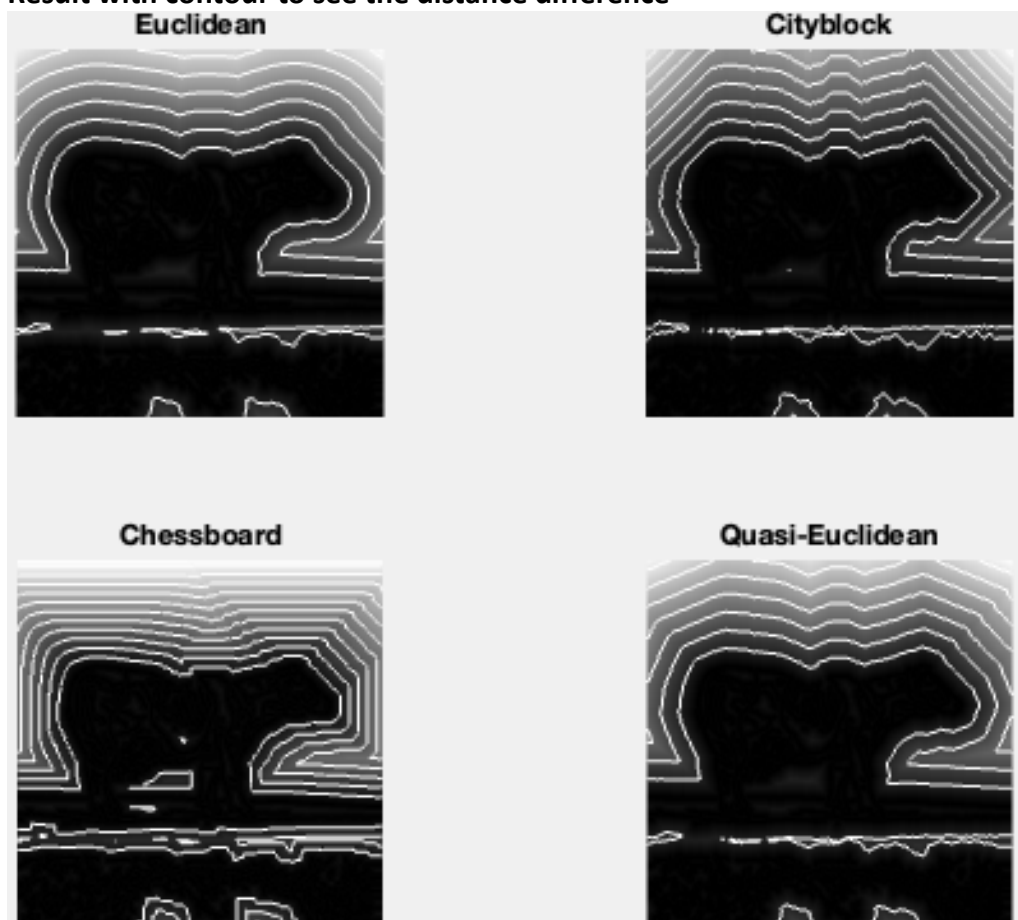


The difference between the four methods is their distance calculations. Euclidean distance will make the pixels spread evenly from the edges so that the result seems more smooth. Cityblock calculation will focus on the 4-connected neighborhood, so the result is more like straight lines. The Chessboard method will focus on the 8-connected neighborhood, so the result is spread evenly to the eight directions as a picture (more detailed). Lastly, the quasi-Euclidean method will measure the total Euclidean distance along with a set of horizontal, vertical, and diagonal line segments, which will make the result more like the combination of Euclidean distance and Chessboard. I think based on the result above, the **Chessboard distance performs the best** since the edge is more specific. However, if we want to **get a smoother result, I think Euclidean distance and Cityblock distance** will be better choices.

Description	Illustration	
The Euclidean distance is the straight-line distance between two pixels.	 Image	 Distance Transform
The city block distance metric measures the path between the pixels based on a 4-connected neighborhood. Pixels whose edges touch are 1 unit apart; pixels diagonally touching are 2 units apart.	 Image	 Distance Transform
The chessboard distance metric measures the path between the pixels based on an 8-connected neighborhood. Pixels whose edges or corners touch are 1 unit apart.	 Image	 Distance Transform
The quasi-Euclidean metric measures the total Euclidean distance along a set of horizontal, vertical, and diagonal line segments.	 Image	 Distance Transform

(Cited from <https://www.mathworks.com/help/images/distance-transform-of-a-binary-image.html>)

Result with contour to see the distance difference



(4) Issue

The issue I encounter in this problem is to deal with the pixels that will not be covered by the forward and backward process, which we possess a value of Inf in the result. I eventually calculated those values and manually feed them with the results of their neighbor so that the whole result will be correct.

(5) Surprise

Distance transform is an interesting method since we could segment the object based on their edges and draw the contours with their distance from the edges.

2.2 Chamfer Matching

(1) What is the minimum Chamfer distance for the given template?

My minimum Chamfer distance is 1 since the ratio would be 1 if all pixels are correct in their position.

 minDist

1

(2) Is there a better way to search for the minimum chamfer distance rather than doing an exhaustive search? Suggest an alternative.

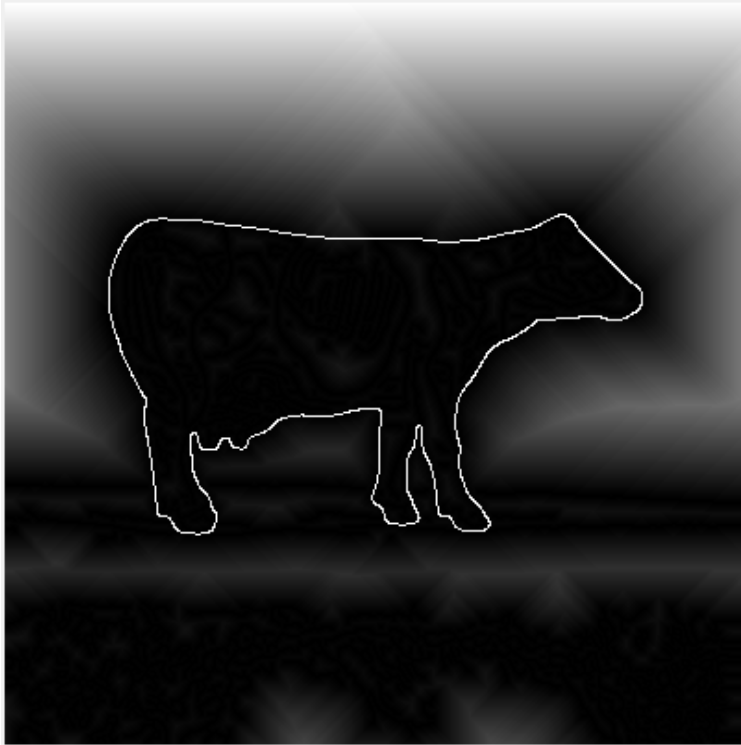
I considered that using a matrix to sum up all pixels will be more time and space-efficient than doing the exhaustive search. We can use the matrix to fit on our target and operate it as a sliding window to sum up the total distance in each iteration, and eventually record the minimum one.

(3) Display the final image of the cow with the template correctly superimposed on it.

cow.png + template.png (highlighted as red line)



cow.png after using canny edge detection and distance transform + template.png



(4) Issue

The issue I encounter in this problem is to figure out how to Chamfer Matching and how to calculate the minimum distance of it. I solved it by watching so tutorial videos on YouTube and finally figuring out an algorithm by myself.

(5) Surprise

I am amazed by the ability of the algorithm that it could perfectly fit the edge onto our target image. Although the algorithm may seem simple, the overall result is very amazing. It performs exactly like object detection.

(Graduate Credit) Fast Directional Chamfer Matching

(1) Read the attached paper on “Fast Directional Chamfer Matching” and write a summary of your understanding in about a page or two.

Paper Cited: <https://www.merl.com/publications/docs/TR2010-045.pdf>

Introduction: In the paper, they improve the accuracy of chamfer matching while reducing the computational time from linear to sublinear. The defect of origin Chamfer Matching => 1. If the original image has a messy background, the result would not be that good. 2. Large computational cost to find out the matching position and need to increase the robustness.

Contribution of the Paper:

1. **Add in edge orientation information in the matching algorithm (like seeing if the template.png is matched in cow.png)** such that the resulting cost function is piecewise smooth, and the cost variation is tightly bounded.
2. **Create a sublinear time algorithm for the directional chamfer matching score** using techniques from 3D distance transforms and directional integral images.
3. **The smooth cost function allows to bound the cost distribution of large neighborhoods and skips the bad hypotheses within.**

Implementation:

(1) **From Chamfer Matching method -> Directional Chamfer Matching method:** In order to add robustness to the original Chamfer Matching method, we include direction difference onto the distance difference of the original algorithm. That is, the new method Directional Chamfer Matching added the direction difference at the calculation of its score formula to increase the robustness.

(2) **From Directional Chamfer Matching method -> Fast Directional Chamfer Matching method:** The best computational complexity for the Chamfer Matching algorithm (without adding orientation term) is $O(n)$, which n stands for the number of template edge points. **So Fast Directional Chamfer Matching method introduces three stages to optimize the time complexity:**

1. Line representation of the template edges

Preprocess the template edges into m number of straight lines connected. Then, the algorithm **hypothesizes lines by selecting a small subset of points and their directions**. While the procedure is being iterated, **the set reduces finding a larger amount of feature points**. Moreover, the noise is filtered, and **the directions are more precise**.

2. New sublinear time algorithm -> Three-Dimensional distance transform representation (DT3)

The three-dimensional here is to add a third discrete angle dimension onto the original XY dimension. That is, to split the space of $[0, \pi)$ into q sections, for example, into $0^\circ, 22.5^\circ, 45^\circ, 67.5^\circ, 90^\circ, 112.5^\circ, 135^\circ, 157.5^\circ$ 8 directions and calculate the DT3 value. Since the original algorithm for the matching score consumes too much time (the time complexity nearly $O(\text{row} * \text{column})$, in which the row and column is the target image's row and column we want to fit in), DT3 is used to compute the matching cost in sublinear time. The operations are to first quantize the edges into discrete orientation channels. Then, a two-dimensional distance transform is computed for each orientation channel. Lastly, DT3 is computed and integrated along the discrete edge orientations.

3. Directional integral image representation over distance transforms

Lastly, a **tensor of integral distance transform (IDT3) is computed to evaluate the summation of costs over any line segment in $O(1)$ operations**. A hypothesis is eliminated during the summation if the cost is larger than the detection threshold or current best hypothesis. Therefore, only a few arithmetic operations are performed on most of the hypotheses.

Conclusion:

Fast directional chamfer matching algorithm shows good performance in object detection & localization, human pose estimation, and 3D pose estimation. Its experiments show that the

proposed approach improves the speed of the original chamfer matching up to an order of 45x, and it is much faster than many states of art techniques while the accuracy is comparable.