Assignment 11: Shape Finder
CS 730/830, Spring 2025
Due at **11pm on Mon, Apr 21**

Overview

You will write a program to find objects in simulated 2D laser rangefinder (LIDAR) data. Undergraduates will find line segments. Graduate students will extend their program to also find circles. Your program will be given data points on standard input. There may be significant noise. Your program will emit the set of lines (and for grad students, circles) that it found in the image. We recommend that you use the RANSAC algorithm. We provide an problem generator and a converter from the vacuum world / RRT map format so that you can play around and tune your implementation's parameters to achieve good performance. Hints on fitting data can be found in the Appendix at the end of this document.

Input/Output

The data are given in ASCII format. The first line specifies the height of the world, the second line the width. Subsequent lines are pairs of x and y coordinates. You can assume that the robot's sensor has an accuracy of approximately 1, so 'adjacent' data points will be around that far apart. Lines starting with `/` are comment lines and should be ignored.

Your program must emit all the circles it finds (zero of them, in the case of undergrads) and then all the line segments it finds. Circles are specified by center $\langle x, y\rangle$, and radius, lines by the $\langle x, y\rangle$ coordinates of the endpoints. As in:

```
number of circles: 1
34.1 56.7 5.0
number of lines: 2
34.1 45.5 162.6 73.2
10.8 23.3 72.5 116.1
```

The coordinate system has $\langle 0, 0\rangle$ in the bottom left corner.

Supplied Utilities

We supply:

`ransac-reference` a sample solution (doesn't find circles).

`ransac-harness` a test harness to evaluate your program. It takes your program as a command-line argument and the data on standard input. Runs your program on the data and then draws a picture of the resulting fit to a PDF file. `-noviz` deactivates the PDF output, `-o` *`filename`* controls where the PDF is written (default is `foo.pdf`).

`image-generator` generates sample data. Command line arguments include `-he` height `-wi` width `-l` number-of-lines `-c` number-of-circles and `-n` number of noise cells. Try `-h` for details.

`txt-to-pts` converters ASCII images to a list of points. On standard input, expects an image in a format similar to the vacuum world (assignments 1 and 2) and motion planning (assignment 3) maps. All cells will be either empty (`_`) or blocked (`#`). Emits points to standard output.

Write-up

Electronically submit your solution using the instructions on the course web page, including your source code as well as some pictures from your program running with the visualizer and brief answers to the following questions:

1. Describe any implementation choices you made that you felt were important. If you implement anything beyond the assignment as written, please be sure to discuss it. Clearly explain any aspects of your program that aren't working. Mention anything else that we should know when evaluating your work.
2. What sorts of input will cause your program to fail? In other words, what does it have trouble with?
3. What suggestions do you have for improving this assignment in the future?

Design Suggestions

Until your data fitting code works, try images that contain only one simple shape.

Appendix: Fitting Data

You'll be dealing with numbers, so don't forget to watch for things like division by 0. You should usually be able to deal with this by choosing new random points.

1. You will find yourself wanting to fit a line to two or more points. Many line fitting methods assume that, for particular known values of x, you have measured some y values which might be corrupted by noise. Hence they find the line that minimizes the squared error in y at each x value. But in our situation, both the x and y values for a particular measurement could be subject to noise. We want to minimize the distance of each point from the fitted line, and this distance should be measured by the shortest distance to the line from the point (not just the distance in the y direction).

The most robust way to do this is to use polar coordinates, but that gets complicated. In the name of simplicity, feel free to just use the equations for orthogonal regression on Wikipedia: the 'Solution' section under 'Deming regression,' with $\delta = 1$. Note that $\hat{\beta}_1$ is the slope of the line you want and $\hat{\beta}_0$ is its y-intercept.

2. Given a line $y = ax + b$, you will want to compute the distance from a point (x, y) to the nearest point lying on that line. This is $|ax - y + b|/\sqrt{a^2 + 1}$.

3. Now that you are in the midst of the assignment, you are probably realizing that what you really want is to fit a line *segment* to your data, not an infinite line. To extract a segment from your line:

a) Project the inliers to lie along the line itself. For each point, the formula for the x value is given in the 'Deming regression' article as $\hat{x}_i^*$ right after the line equations. Use the line to get the y value for the point. Now that the points are on the line, we can represent segments by their start and end points.

b) Sort the points (for example, by their x values) and walk along them, splitting the line into segments whenever there is a large enough gap between points (say, for example, a distance of 4.0).

c) Among all the resulting segments that have a sufficient number of inliers (say 7) and a sufficient density of inliers (say 0.7 inliers per unit length), call the one that explains the most inliers the best. If there are no such segments, then this line fit fails.

4. The RANSAC pseudocode from lecture recommends trying many fits and choosing the best. The value of 'many' might be 500.

5. Grad students will also need to fit circles. Find the circles before looking for lines. To find the center (x, y) and radius r of a circle from three points (x_i, y_i):

$$\begin{aligned}
m_a &= \frac{y_2 - y_1}{x_2 - x_1} \\
m_b &= \frac{y_3 - y_2}{x_3 - x_2} \\
x &= \frac{m_a m_b (y_1 - y_3) + m_b (x_1 + x_2) - m_a (x_2 + x_3)}{2(m_b - m_a)} \\
y &= -\frac{1}{m_a}(x - \frac{x_1 + x_2}{2}) + \frac{y_1 + y_2}{2} \\
r &= \sqrt{(x_1 - x)^2 + (y_1 - y)^2}
\end{aligned}$$

You probably want to enforce a minimum radius. Don't worry about refitting the circle to all the inliers. Finding the distance of a point from the circle is easy: find its distance from the center and compare it to the radius.

6. Feel free to tweak stuff to improve performance, but be sure to test on several different problems to avoid 'overfitting' your algorithm!