Gauntlet

Tuesday, May 4, 2021 11:30 PM

gauntlet

Chapter 30

The Gauntlet

30.1 Overview

You have mastered the Bridge of Doom and navigated Flatland. Now you face the most challenging challenge you've ever been challenged with. In this final challenge, you will help your robot to dodge through obstacles on your way to the ultimate prize – knowledge.

Throughout this semester, you have applied many quantitative engineering analysis tools. For this challenge, you'll use a powerful new sensor (the laser scanner), explore some algorithms for optimization, and use the mathematics of potential functions and vector fields.

If you'd like you can work in a group (up to 3 people total per group) to complete the challenge and the write-up.

Learning Objectives

This challenge has varying levels of difficulty. The learning objectives vary based on the level you choose.

1. Fit models of lines and circles to laser scan data (level 2 and 3).
2. Implement outlier resistant optimization via the RANSAC algorithm (level 2 and 3).
3. Create suitable potential functions and vector fields to guid a robot to a goal while avoiding obstacles (all levels)
4. Transforming between multiple coordinate systems (all levels)

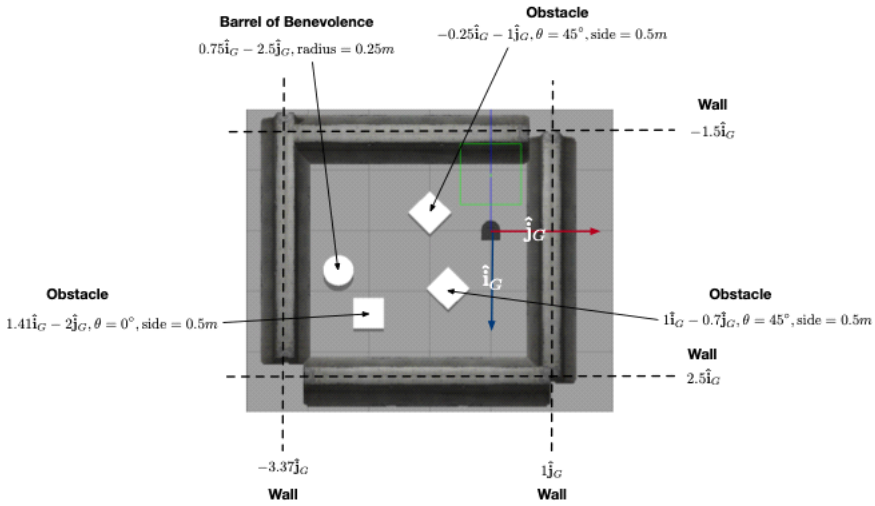


Figure 30.1: A top-down view of The Gauntlet. You should write a program to guide the Neato past the obstacles (boxes) to the goal (the Barrel of Benevolence).

30.2 The Challenge

Your goal is to write a program to pilot your (virtual) Neato through a series of obstacles (concrete barriers and boxes). You will detect these obstacles using your robot's onboard laser scanner, which can detect obstacles within a 5m range (see Figure 30.1).

This final QEA-I challenge has a hierarchy of missions, which get progressively more difficult. For all missions, the goal is to gently tap the Barrel of Benevolence (BoB). For all missions, the Neato starts at coordinate $0\ m\hat{\mathbf{i}} + 0\ m\hat{\mathbf{j}}$, facing along the $+\hat{\mathbf{i}}$ direction.

Here are the conditions for the missions:

Level 1 You are given the coordinates of the BoB and the obstacle and wall locations (see Figure 30.1). You will use these locations to create a potential field and use gradient descent to move your robot through the Gauntlet to the BoB. The path can be entirely pre-planned.

Level 2 You tackle the same challenge as level 1, but you must use the LIDAR to detect and avoid obstacles (i.e., you should not use the obstacle and wall locations in Figure 30.1, but instead determine them from the robot's sensors). You can plan your entire path based on your initial LIDAR scan or dynamically update your path as you go. The location of the BoB is known. To tackle this level of the challenge, you will need the RANSAC material in Chapter 31.

Level 3 You tackle the same challenge as level 2, but you are given the radius of the BoB (see Figure 30.1), but not the coordinates (i.e., you must use LIDAR scans to identify and locate the BoB). You can plan your entire path based on your initial LIDAR scan or dynamically update your path as you go. To identify the BoB, you will need to differentiate the distinctive, circular shape of the goal from the linear shape of the obstacles. If you choose this mission, we recommend you look at the circle-fitting extension in Chapter 32.

To achieve whichever mission(s) you choose to accept, you will apply what you have learned about potential functions and vector fields. You may also choose to extend any of the above missions by applying or creating another goal-seeking/obstacle-avoiding algorithm, not taking the BoB's radius as an input, or trying to reach the target as soon as possible.

30.2.1 Loading the Gauntlet

To start the Gauntlet world, you can use the `qeasim` script by running the following script in MATLAB.

```
qeasim start gauntlet_final
```

30.3 Deliverables

Don't think of these deliverables as only items to check off, but rather think of them as scaffolding to get maximum learning from the task at hand. By creating these intermediate deliverables, you will be able to more easily debug your code as well as your understanding of the algorithms you are utilizing.

Note that all of these deliverables can be done with your group (if you are working with others for the challenge). Make sure to clearly specify who is in your group for the challenge on each deliverable. Unless otherwise noted, the components are required for all levels of the challenge.

Mapping and Path Planning (due 5/6, but suggested completion date is 5/3) [6 pts]: We suggest that you make significant progress on the math side of this project before class on Monday. Create a document with the following plots:

1. A map of the Gauntlet using LIDAR scan data. Please see details of this map for each level of challenge. [2 pts]
   (a) **Level 1:** The map should be an ensemble of data points from laser scans collected at various positions around the Gauntlet and translated/rotated to the global coordinate frame. The points from each scan should be shown in a different color, and all walls, obstacles, and the BoB should be captured by the scans. The plot should have appropriate labels and units.
   (b) **Level 2:** A map of the Gauntlet created using laser scan data as above, and with walls and obstacles fitted as line segments using your RANSAC algorithm.
   (c) **Level 3:** A map with fitted lines as above, with the addition of a fit for the BoB.
2. An equation and a contour (or 3-D) plot of the potential field you developed for the pen. [2 pts]
   (a) **Level 1:** The position of the walls, obstacles, and BoB can be based on the positions of the features shown in Figure 30.1.
   (b) **Level 2 and 3:** The potential field should be generated based on the locations of obstacles and the BoB identified using the laser scans and line/circle fitting algorithms.
3. A quiver plot of the gradient of your potential field. [1 pt]
4. A path of gradient descent from the starting point to the BoB. [1 pt]

These plots should be clearly labeled and readable (decent size fonts!). **You will turn these in on Canvas through the Mapping and Path Planning assignment.**

Navigating the Gauntlet (due 5/6) [8 pts]: Prepare a video and a **brief** writeup of your work on this challenge.

Your final writeup should center around critical information and informative figures. It can be as short as you want, but it must contain the following components:

1. A short introduction stating the mission you chose and a brief summary of the strategy you used to solve the challenge. You might find it helpful to refer to the decomposition exercise from Week 7. [1 pt]
2. Some experimental data that shows, quantitatively, how well your system worked, plus a few sentences of explanation. Note: you could make use of the collectDataset_sim.m function for this.

(a) The time it took your robot to get to the BoB (for whichever mission you choose) and the distance it traveled. [1 pt]

(b) A plot with: a map of the relevant features of the Gauntlet based on a LIDAR scan (levels 2 and 3) or the features given in Figure 30.1 (level 1), the intended path of gradient descent, and the path your robot took calculated from the wheel encoder data. This should be a legible plot with axis labels, a legend, and a caption...the works. [3 pts]

3. A link to a video of your robot in action. [2 pts]

In addition to the writeup, you should also turn in your (commented and readable!) code. [1 pt]

Chapter 31

Optional Material: The RANSAC Algorithm and Finding Lines

Schedule

Learning Objectives

From LIDAR data, (r, θ), that contains a linear object and outliers,

- Design the steps for a RANSAC algorithm to find a line in a laser scan.
- Create an algorithm to sort the data points that fit the line and those that do not.

Challenge: Find multiple lines using the RANSAC algorithm from a LIDAR data set with multiple linear object signals.

31.1 RANSAC [20 minutes]

Today we'll be learning a technique for optimization in the presence of outliers. The algorithm is called Random Sample Consensus, RANSAC for short, and it has applicability to a wide variety of problems in robotics and computer vision. In The Gauntlet™ we'll be using it to find lines in laser scan data even when multiple structures are present (e.g., multiple walls, lines and circles).

31.1.1 Motivating Example

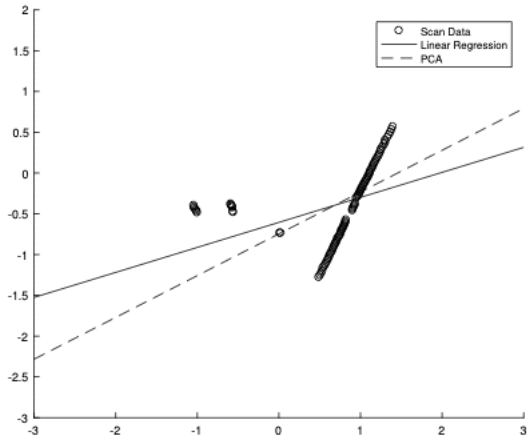


Figure 31.1: The lines of best fit as computed by linear regression and PCA. Due to the mixed structures in the data, the results are very poor.

Previously, you applied line fitting techniques to four different laser scans. The linear regression method worked well in some cases, but it had some clear shortcomings (e.g., when the scan points were oriented vertically). The PCA algorithm was able to overcome some of these limitations, however, there were cases when even PCA failed. For instance, Figure 31.1 shows the results of applying both linear regression and PCA to scan4.mat. Due to the fact that there are outliers (i.e. points that do not lie on the line), both linear regression and PCA find lines that do not correspond at all to the line clearly visible in the scan.

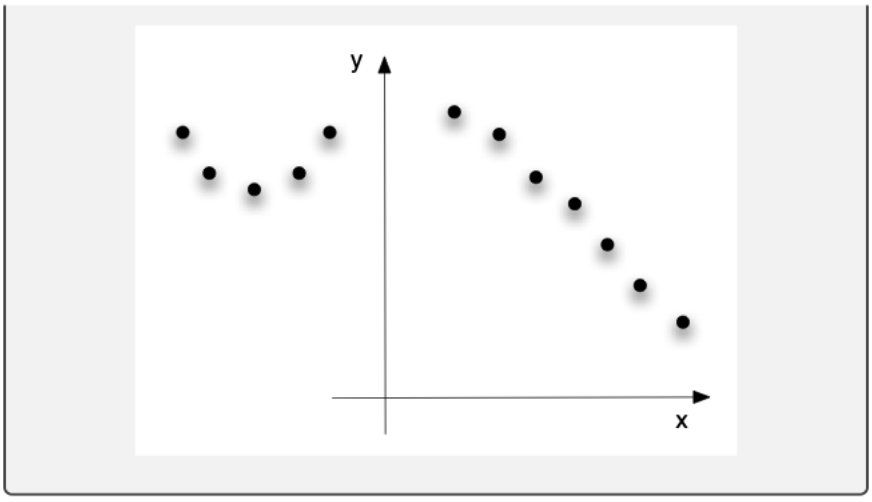
We can conclude from this example that the methods of line fitting that we've learned thus far are effective yet brittle (i.e. they fail when conditions aren't ideal). Motivated by this observation, today we'll be learning how to use the RANSAC algorithm to filter outliers *before* applying one of the line fitting methods you explored in the overnight.

31.1.2 Robust Optimization and the RANSAC Algorithm

The mathematical field of robust optimization provides us with techniques for optimizing functions (such as MSE) that are robust in the presence of outliers. Today, you'll explore a very powerful technique for robust optimization called Random Sample Consensus (or RANSAC for short). The algorithm works by choosing a small subset of data points to fit a model, determining whether or not a significant proportion of the data is consistent with the fitted model, and then repeating this process until a satisfactory model is found. For instance, let's suppose we want to find a line in this laser scan.

Exercise 31.1

1. Qualitatively, how many unique structures do you see in this laser scan?
2. Using the Zoom annotation tools, discuss and draw in your "best fit" line(s).
3. Based on your experience with linear regression and PCA during Day 7, where do you expect these algorithms would place a fit line? Go ahead and sketch them as well.


y
x

The RANSAC algorithm starts by randomly choosing a minimal subset of data points required to define a line (which of course is 2). Given these 2 points, we define our candidate line as the line that passes through both of those points (this is our "model"). Next, we divide *all* of the points in our laser scan into two groups. The first group, called *inliers*, consists of points in the laser scan that are sufficiently close to our candidate line. The second group of points, called *outliers*, consists of points in the laser scan that are far from the candidate line. We are free to choose the notion of what it means to be close or far from our candidate line. Based on our experience in the overnight, the perpendicular distance seems like a reasonable choice to measure the closeness of a point to a line (recall that this is what PCA uses). Therefore, we'll use the perpendicular distance as a metric for deciding whether a point is an inlier or an outlier. Figure 31.2 shows an example of applying this procedure. The inliers are marked in either white (for the two points used to make the line) or blue. The inliers are the points that fall within a perpendicular distance d from the line (d is a threshold value that you specify to RANSAC). The outliers (those that fall outside of a distance d of the line) are marked in red. While the line defined by the two randomly chosen points results in some inliers, the number of inliers (4 including the points used to make the line) is perhaps not as large as it could be.

Exercise 31.2

1. On Figure 31.2, draw a new best fit line and threshold lines that would optimize the number of inliers vs. outliers. Identify which points would serve as the endpoints of this line.
2. How did you choose your line/endpoints? Discuss possible strategies to accomplish this task.

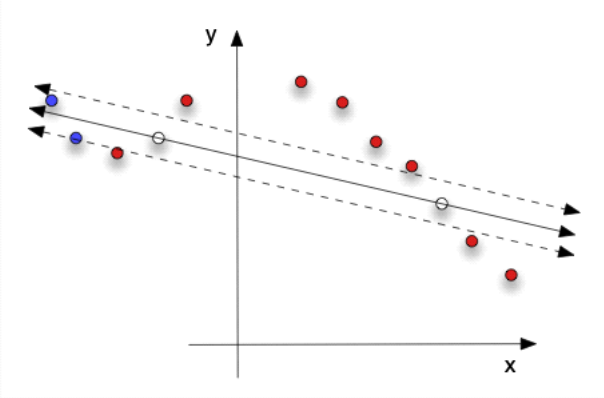


Figure 31.2: The white points represent the randomly chosen subset of two points to define the line. The solid line passing through them is the resultant line. The two dashed lines correspond to the inlier threshold (defined by measuring a specified distance d perpendicularly from the line). The points that are inliers are marked in blue and the outliers are red.

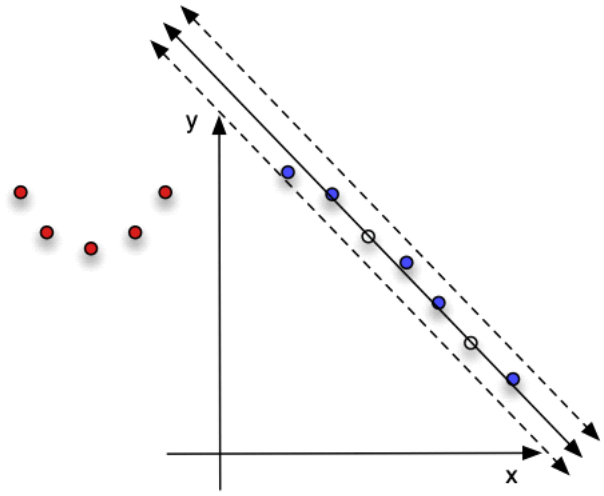


Figure 31.3: The white points represent the randomly chosen subset of points and to define the line. The solid line passing through them is that resultant line. The two dashed lines correspond to the inlier threshold (defined by measuring a specified distance d perpendicularly from the line). The points that are inliers are marked in blue and the outliers are red.

Next, we choose a new random set of 2 points, define the line passing through those points as our candidate line, and determine inliers and outliers. If this new candidate line results in more inliers than the first one we tried, we save the line for later. We repeat the procedure (choosing two random points, determining inliers, and testing to see if the number of inliers is the highest we've found so far) n times, where n is a parameter that you specify to the RANSAC algorithm. If all goes well, one of these n lines results in something like Figure 31.3. For the randomly chosen points shown in the figure, the number of inliers is 7 which looks to be about as good as we can do with this scan data.

As a final consideration, in the case of fitting lines to laser scan data, we want to be able to determine not just where the lines are in the scan, but also where the lines begin and end. This is crucial since the lines we are finding correspond to things like the beginning and end of walls or obstacles. There are a few ways to approach this task, and we'd like you to come up with your own method as part of today's activities. We are certainly here to scaffold this, but wrestling with this a bit will help to build your intuition about the geometry of the problem.

31.2 Coffee Break [10 minutes]

31.3 Converting from Intuition to Pseudocode [45 minutes]

Let's take stock of where we are. You should now have a pretty good idea about how RANSAC works on a conceptual level. Next, you'll be working to take this intuition and convert it into pseudocode.

When writing pseudocode it helps to start by stating your high-level conceptual understanding of how to solve the problem, and then, work towards progressively more concrete statements of how to solve the problem. Motivated by this idea, here's a possible procedure you can use with your partner to generate pseudocode.

1. **Clear up conceptual misunderstandings:** You just read through some text that described the RANSAC algorithm. Were there any parts that didn't make sense at a conceptual level? If so, make sure you work through these with your partner. If you can't figure out one of your questions, let us know! We're here to help.

2. **Simulate the algorithm:** at the whiteboard, simulate the steps that RANSAC would go through to find a line in a laser scan. You can do this through a combination of pictures (like the ones in Figure 31.3) and explanatory text (e.g., select points at random). As you go through this process you may find that you don't understand some of the details as well as you thought. This is another chance to ask for help from the teaching team.

3. **Write out the major steps:** Next, write the sequence of major steps the algorithm should perform. By this point your descriptions should be getting more precise (although not necessarily more detailed). Deciding how big to make each step is a bit of a balancing act. You want to avoid sequences such as "1. Do the thing 2. ??? 3. Profit", but you certainly don't want to have a 25 step process. Shoot for something on the order of 5-7 steps. Each of these steps can later be subdivided into smaller pieces.

4. **Figure out your functions:** the major steps you've defined can be thought of as the functions that you will write to implement your algorithm. For instance, you may have come up with a step called "compute inliers and oultiers". The fact that you identified this as a major step for your algorithm suggests that making a function that performs this computation is probably a good idea. At the whiteboard write out a list of the functions you will create when implementing RANSAC. Make sure to describe what each of these functions expects as input, what it will generate as output, and what it does.

5. **Write your pseudocode:** Next, write pseudocode for each of the functions you've identified as well as pseudocode that stitches these functions together to implement RANSAC. Your pseudocode should be written in natural language (avoid using actual MATLAB syntax in your pseudocode), yet precise enough that there is little ambiguity about how it could be translated into actual MATLAB code. To

better make this last point, here are two different potential ways to write pseudocode for finding the maximum element in a list of values.

```
Input: a list of numbers L
for each number x in the list L
    if x is the highest value so far
        remember x
return the highest value we found
```

This pseudocode has some good properties. It is written in natural language and it specifies the inputs of the function. On the negative side there is still a good deal of ambiguity here. How do I test if "x is the highest value so far"? How do I "remember x"? Here's a version that improves the pseudocode in this respect.

```
Input: a list of numbers L
initialize a variable called maxval to the value negative infinity
for each number x in the list L
    if x is greater than maxval
        assign the value of x to maxval
return the variable maxval
```

This new version of the pseudocode can be unambiguously translated to a computer program.

Exercise 31.3

Write pseudocode to implement the algorithm described in Section 31.1.2. Your pseudocode should be capable of going from a polar coordinate representation of a laser scan to the endpoints, in Cartesian space, of a line segment that corresponds to a line in the laser scan.

31.4 Implementing Your Algorithm [50 minutes]

In this section you'll be translating your pseudo-code into MATLAB.

Exercise 31.4

Implement RANSAC using the pseudocode you wrote in the previous exercise. Test your algorithm on the data in scan4.mat. As a suggestion, you should define a top-level function called *robustLineFit* that takes as input a polar representation of the laser scan, the threshold d to use to determine whether a scan point is an inlier, and n the number of random lines to try. Once you've implemented your algorithm, experiment with d and n to understand their effect.
Debugging and implementation tips:

1. Visualize, visualize, visualize. For instance, make sure your procedure for determining inliers is correct, you should plot the fitted line, the inliers, and the outliers. Make sure to use different colors to plot the inliers versus outliers.
2. Develop incrementally. Build your program bit by bit. Experiment in the command window before writing code in your MATLAB script.
3. Set breakpoints. This can be accomplished by either using the *keyboard* statement or by creating a stop sign by clicking to the left of the line of MATLAB code. These breakpoints

are particularly useful in two situations. The first situation is obvious – when attempting to debug code. The second situation is when you are about to write an intricate section of code. In this case, set the breakpoint where the code will eventually go. Run your existing (but incomplete) code. When MATLAB stops at the break point, you can prototype your solution in the command window before adding it into your script.

Recommended Extension: If all has gone well, you should have a beautiful line segment fit to the data in *scan4.mat*. Unfortunately for you, The Gauntlet™ is a bit more complicated than the Chamber of Emptiness™. Next you'll be updating your code to find multiple lines.

Exercise 31.5

First, write pseudocode to find multiple lines in a laser scan. When doing this you should HEAVILY leverage what you did in exercise 3. For instance, if you have a function called *robustLineFit* which takes a scan and computes the best fitting line segment, you can call it repeatedly to find multiple lines. Specifically, each time you call your *robustLineFit* code, you'll determine a line segment in scan. After determining a line segment, you should remove the inlier points for that line segment from the scan data. By removing these scan points each subsequent line you find will consist of points that were outliers with respect to the lines you previously found. Implement your approach and try it on the data stored in playpensample.mat.
Debugging and implementation tips:

1. In order to write this in a sane fashion, your functions will need to be solid. Make sure you are confident in each function before building on it.
2. You may consider modifying your *robustLineFit* to return multiple pieces of information (instead of just the line segment of best fit). For instance, if you return the inlier points and outliers points separately your life will be a lot easier. If you're not familiar with returning multiple outputs from a MATLAB function, see this page.

Challenge Problem: If all went well you should now be finding all the walls in a fairly complex laser scan. You may have noticed some warts in the lines that your code finds. The biggest issue is that the line segments that are found sometimes have large gaps in them. This causes you to find lines that span across gaps in the environment. Later in the challenge you'll be using these line segments to determine a path through the environment, so it is important that the gaps are not covered over by spurious lines.

Exercise 31.6

Modify your *robustLineFit* code to avoid fitting line segments with large gaps. To do this, define a procedure for computing the largest gap in a candidate line segment. Start out by working on the board to build intuition, and only when you have a good sense of how to solve the problem, implement things in MATLAB.

Solution 31.4

An example solution using the RANSAC algorithm with d=0.1m and n=30 iterations is shown in Figure 31.4. A heavily commented, working RANSAC function is here.

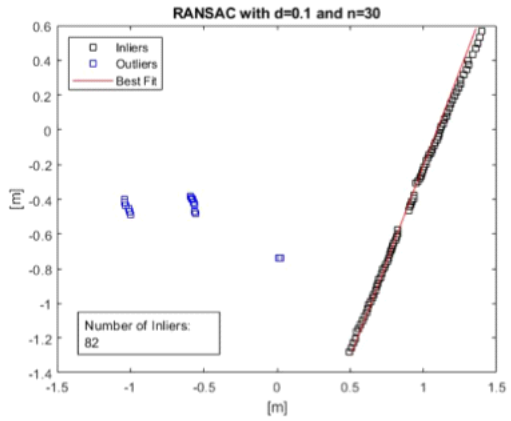


Figure 31.4: Best fit to Scan4 using RANSAC with d=0.1m and n=30.

Solution 31.5

A solution is here.

Solution 31.6

In the solution here, gaps are identified using the code:

```
%we also want to check that there are no big gaps in our walls. To do
  %this, we are first taking the distance of each inlier away from an
  %endpoint (diffs) and projecting onto the best fit direction. We then
  %sort these from smallest to largest and take difference to find the
  %spacing between adjacent points. We then identify the maximum gap.
   biggestGap = max(diff(sort(diffs(inliers,:)*v/norm(v))));
```

Chapter 32

Optional Material: the Best-Fit Circle

If you plan to identify and locate the Bucket of Benevolence in The Gauntlet challenge, you will need to be able to find and fit a circle. These steps will guide you through the process of creating a circle-fitting algorithm of your own design.

Exercise 32.1

Assume that you have collected a set of N data points (x_i, y_i). There are multiple approaches to fitting a circle to this data, but we would like you to utilize the tools you have learned in QEA this semester, specifically linear regression and the RANSAC algorithm.

1. What are the different ways of describing a circle? Think about explicit, implicit, and parametric representations. Don't forget that there are various ways of describing a circle using each of these representations.
2. We would like you to set up the circle fitting problem using linear regression techniques from linear algebra i.e. an over-constrained system of linear equations where A*w = b. Choose one of the circle equations you identified and write out the A, w, and b matrices.
3. How would you solve this system?
4. Now imagine that you wanted to fit a circle using something more like a RANSAC approach. Describe the algorithm you would use, assuming that you knew the radius of the circle your were fitting (a la BoB). How many points would you need to select to fit each candidate circle?
5. What if you didn't know the radius? How would you have to modify your RANSAC approach?

Exercise 32.2

At this point, you have a couple of approaches in mind, and you've thought through the details of each. Let's test your ideas now by implementing in MATLAB.

1. Write pseudocode for finding the best-fit circle using one of your methods.
2. Generate some test points that lie on a circle of radius 2. Rather than generating points on the entire circle, let's generate points on an arc of 120 degrees to mimic the extent of the data we might get from the NEATO.

3. Test your methods on this data. How good is your best-fit circle? Do your results depend on which method you use, or how you initialize the method?
4. Now incorporate a little noise into your test data set by perturbing the radius of each of your data points (consider using the MATLAB function *randn* to do this). How does this affect your results? Make sure to try your code a couple of times to get a representative picture of how well it works on different noisy data samples.
5. Now implement the pseudocode you generated above into a working RANSAC-style circle fitting algorithm in Matlab. Test your code on the file playpensample.mat which has one instance of the BOB with r=0.1329m.