Problem 1

scissors(imIn, seedRow, seedCol,destRow,destCol):

- Usage: works as specified.
- Optimization: when generating the distances, the diagonal edges can be computed by:

1. kernel = [1 0;0 -1], [0 1; -1 0]; [1 1; 0 0; -1 -1], [1 0 -1;1 0 -1]
2. Do a convolution with imIn and kernel, and do some row shifting.

after that, is a standard shortest path algorithm from seed to destination.

activescissors(imIn):

- Usage:

1. input the image
2. the image will show, **click on the seed** you want, then **press enter**.
3. activescissors will calculate the shortest path tree.
4. while you are happy, **click on any point** on the image, then **press enter**. After that, activescissors will show the cut from the seed to your point.

- Optimization:

- Once the seed is decided, then the shortest path tree is generated, all the shortest pathes can be shown in O(length of path) time. I think this is convinient for testing, so I made this function.

- This works out pretty well.

Problem 2.2

Usage:

- yourcellvar = myHoughCircleTrain(imBW,c,ptlist)
- Outputs a cell object containing the trained data.
- centers = myHoughCircleTest(imBWnew,yourcellvar)
- Outputs the coordinates of the top two detected centers. and also shows them on the screen.

Training stage:

- In lecture 5, the generalized hough transform used the local gradient too decide where to vote. So in the training stage, I do:
- For each point in the point list:

1. Put its **normalized local gradient** into yourcellvar.
2. Put its **relative position** to the reference point into yourcellvar.

- In the detection stage, I do:

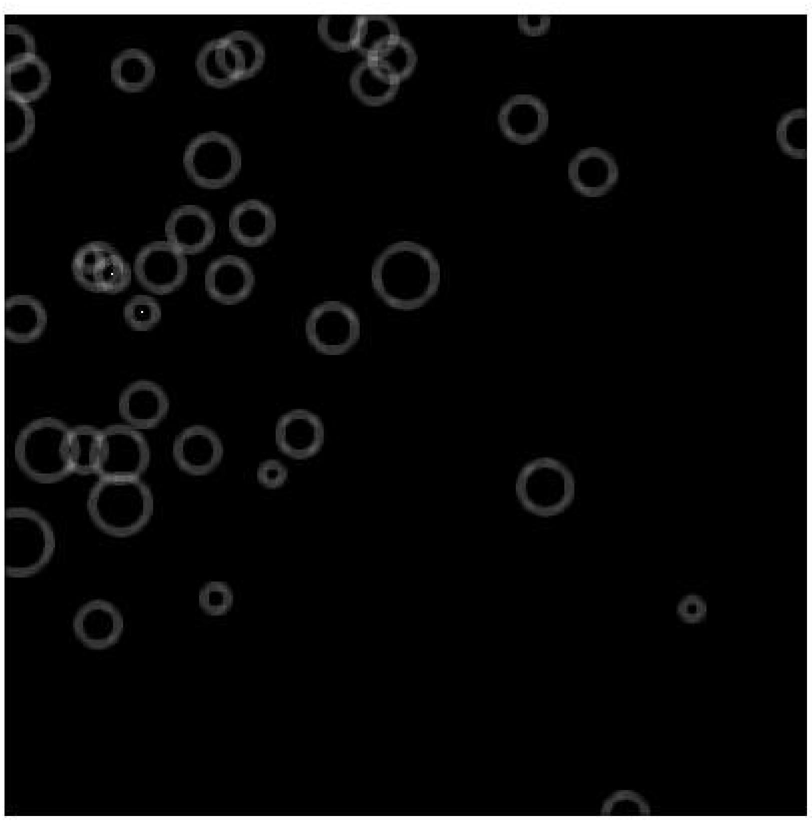
1. Calculate the normalized gradient for each pixel.
2. Find the point with closest gradient in yourcellvar. Since the gradient is normalized, the inner product would be enough.
3. vote, with the -(relative position) of the point in yourcellvar to the pixel.

Failures:

- This training does not do well on scaling. It can do well on test.png, where the circles are about the same size of the training circle, but not the others.

Result image:

- The Image is smoothed to calculate gradient.



  -

Problem 3

Usage:

- mySnake(imIn, imInitial,alpha,beta) : as specified.
- myGSnake(imIn,alpha,beta) :

1. Image shows.
2. Click on the points on the snake.
3. Press enter.

implementation:

- Dynamic programing to find the next best snake.
- Basically same algorithm as class. But two differences:

1. It is in a loop.
2. It has second order terms.

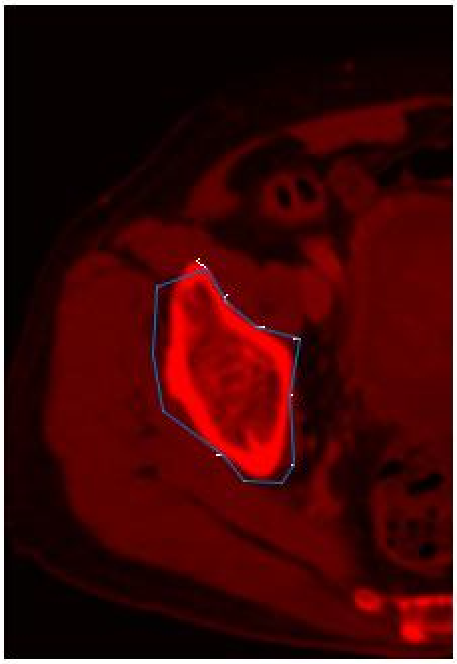
- To fix the algorithm to work on this problem, you have to do the following:

1. Keep track of all the best configurations by fixing the previous 2 points. i.e. you will have 81 previous states instead of 9.
2. To make it a loop, you have to fix the first point and second point, and find the corresponding 81 snakes. (If you only have the first order term, then you only fix the first point, and find all 9 snakes).

3. The added Energy function for the last point is more complex. It interacts back to the first and second point.

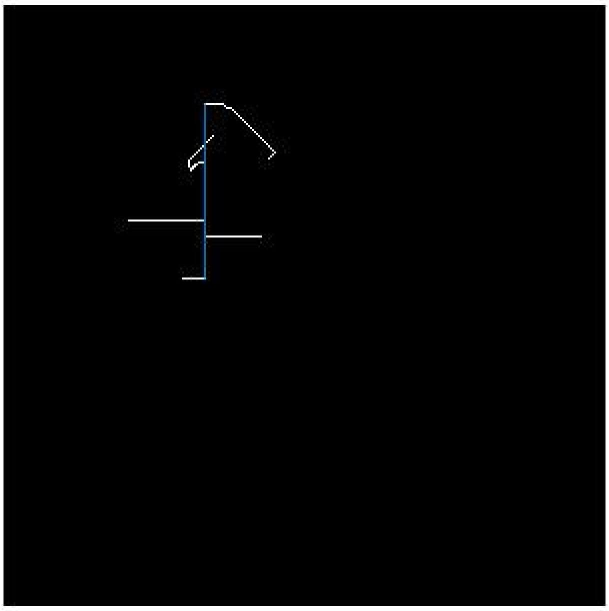
Results:

- When first used, the internal energy was donminating. I had to fix alpha and beta to a very small value.
- I thought good alpha and beta is about the value of the **average gradient of the gradient of the image**, divided by your **average edge length** on the snake. Because you have square terms, the change in internal energy when moving a vertax is about the edge length. But alpha needs to be even smaller.
- Test image with alpha = 0.000001, beta = 0.0000001, 15 iterations.



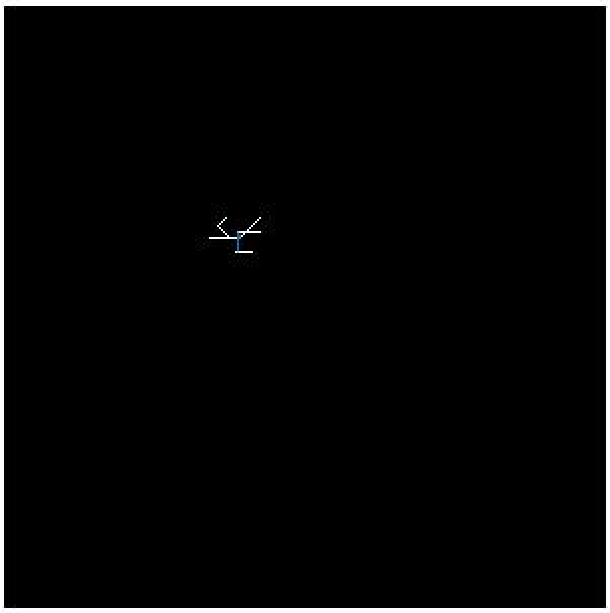
  -

- Test image on the second order term: alpha = 0, beta = 1, image is all black.
- The snake evolves into a straight line.



  -

- Test image on the first order term: alpha = 1, beta = 0, image is all black.
- The snake evolves into a point.



  -