Access the code here: https://github.com/nazelif/Scout_Rover

NOTES:
I dated every log entry, with more recent entries on top.
Almost all the bugs I encountered are mentioned here. Resolved ones marked as “Solved” (most of the time)
The To-Do list items below got updates as I made progress, and how I solved them are written out under relevant days.

ULTIMATE TO-DO LIST

- **PATH FINDER**
    - **Cost3** averaging is not working well. See notes from 4/24 for an explanation and screenshots of cases. Once it works, compare results of cost2 and cost3 to decide which one is working better. **Currently, we find path on cost2 matrix only.**
    - **Updating the cost function:** Should we make it dependent on the degree to turn?
    - **Size of the rover to path-finding.**

- **LIDAR INPUT**
    - **Subtracting a plane**
    - **The size of each pixel:** a brute way of doing it is to bind them so that each pixel is the size of the rover
    - **Size of the input matrix:** How are we going to find the maximum x and y points in the real case? Should I run a script to find the max points?

- **MOTORS**
    - **Checking if we are at the correct point:** After every point we “descend” or “turn more than x degrees” we expect a high chance of disorientation. Make the cost function depend on that or make another cost function→ then make checks at these points (small lidar scans)
    - **Curving** function does in fact make turning easier, but it doesn’t guarantee that we are avoiding obstacles still :(A possible solution is to incorporate path generation into cost_matrix function… which is tedious :/
    - **Initial orientation:** how are we going to know where we are facing in the real case, in terms of the map

- **HOUSEKEEPING**
    - **What LIDAR will feed into?** In the real case, we won’t generate a terrain
    - Move the code to a Pi
    - Document what each file does
    - Make a bash script to run the code start to finish
    - Make everything in metric… what is 14x14 inches = 0.3556m x 0.3556 m
    - Write details for the **README** for github repository.
    - Write a script to display the path on the DEM
    - **Size of the rover changed= prev 14 inches, now 9x10 inches.**

1. **Crop out 2m up things in z**
2. **Plot the remainder in x and y - done**
3. **How to convert it to a dem**

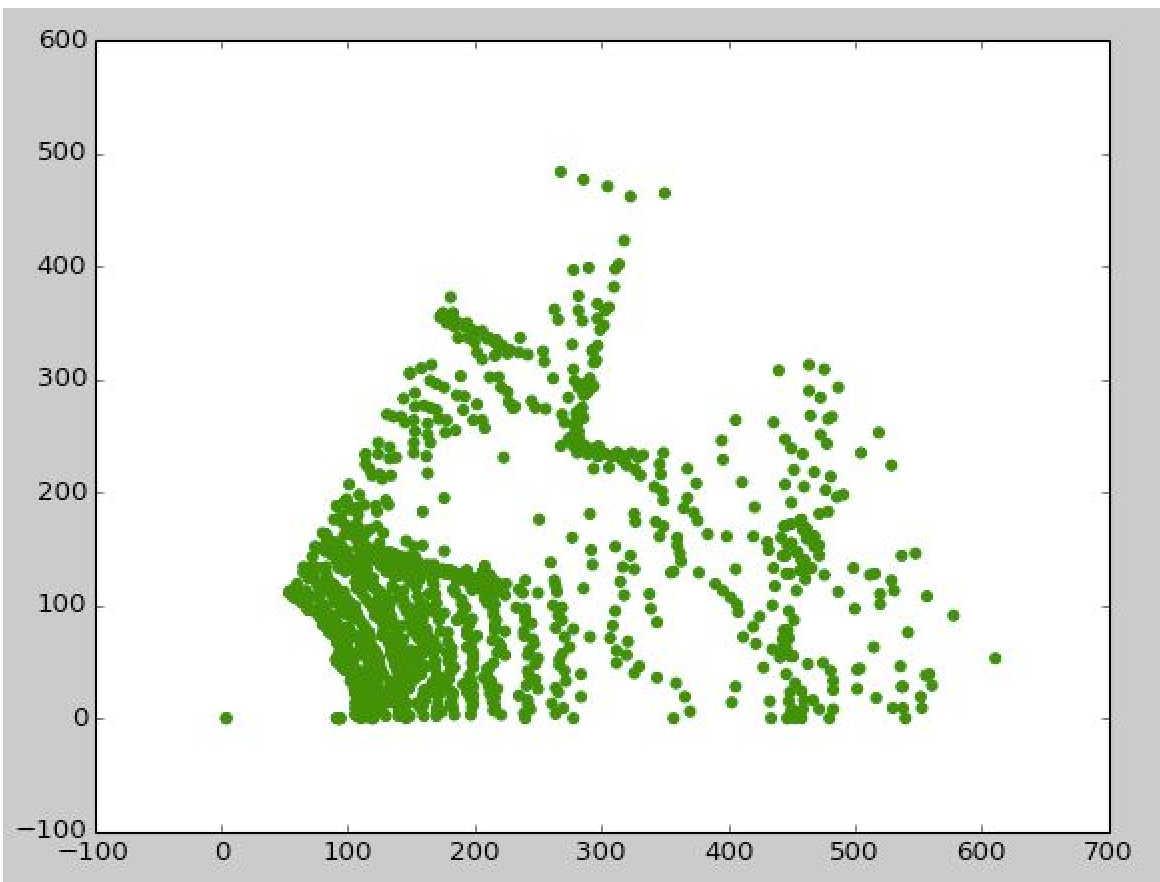
WEDNESDAY 5/16

Today I did the interpolation w Wes.

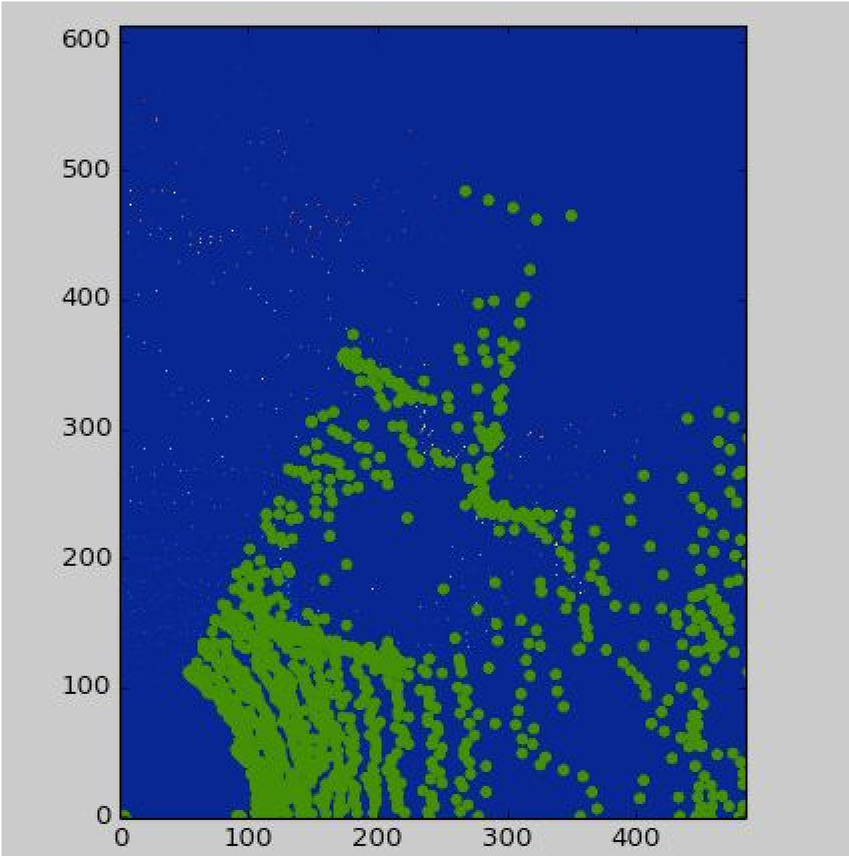
I shifted all my data points from - values to 0 with this code:

```
for i in range(len(x_matrix)):
        x_matrix[i] = x_matrix[i] - x_min
        y_matrix[i] = y_matrix[i] - y_min
```

Note that this preserves the order, so we are fine.
I also cut values where the z is greater than 2 meters aka the trees.
This way we were able to plot a scatter point of the point cloud data.



Next, we create the lidar_result file. When plotted, it looked like a bunch of scattered points. When plotted together with the point cloud data, this looked a bit concerning, because we didn't have all these points plotted with the lidar. Some points went missing...

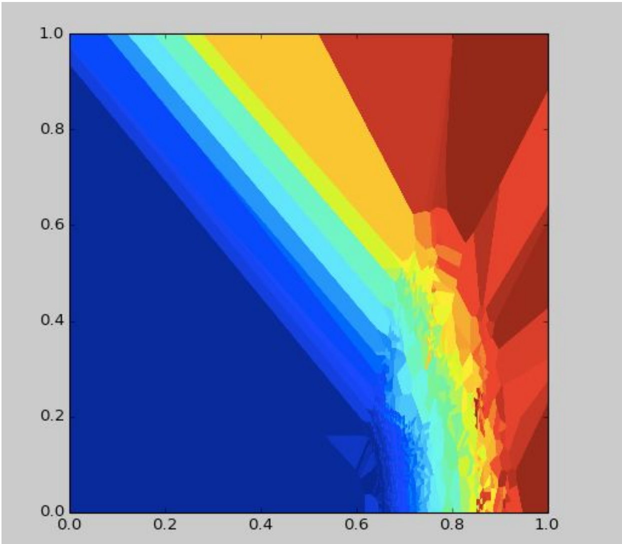


Next came the interpolation step. Figuring out the shapes of matrices took a bit of time, but we managed it at the end.

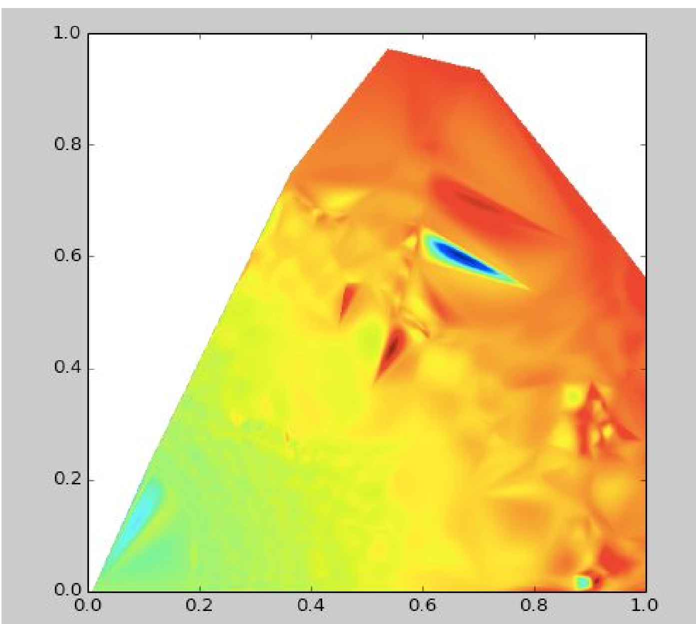
This is the code that went into it

```
grid_x, grid_y = np.mgrid[-1000:700:1, 0:700:1]
#points = arr #600, 2 (just the x and ymatrices)
points = np.asarray([x_matrix,y_matrix]).T
values = np.asarray(z_matrix)
print points.shape
print values.shape
print type((grid_x, grid_y))
grid_z0 = griddata(points, values, (grid_x, grid_y), method='nearest')
plt.imshow(grid_z0.T, extent=(0,1,0,1), origin='lower')
```

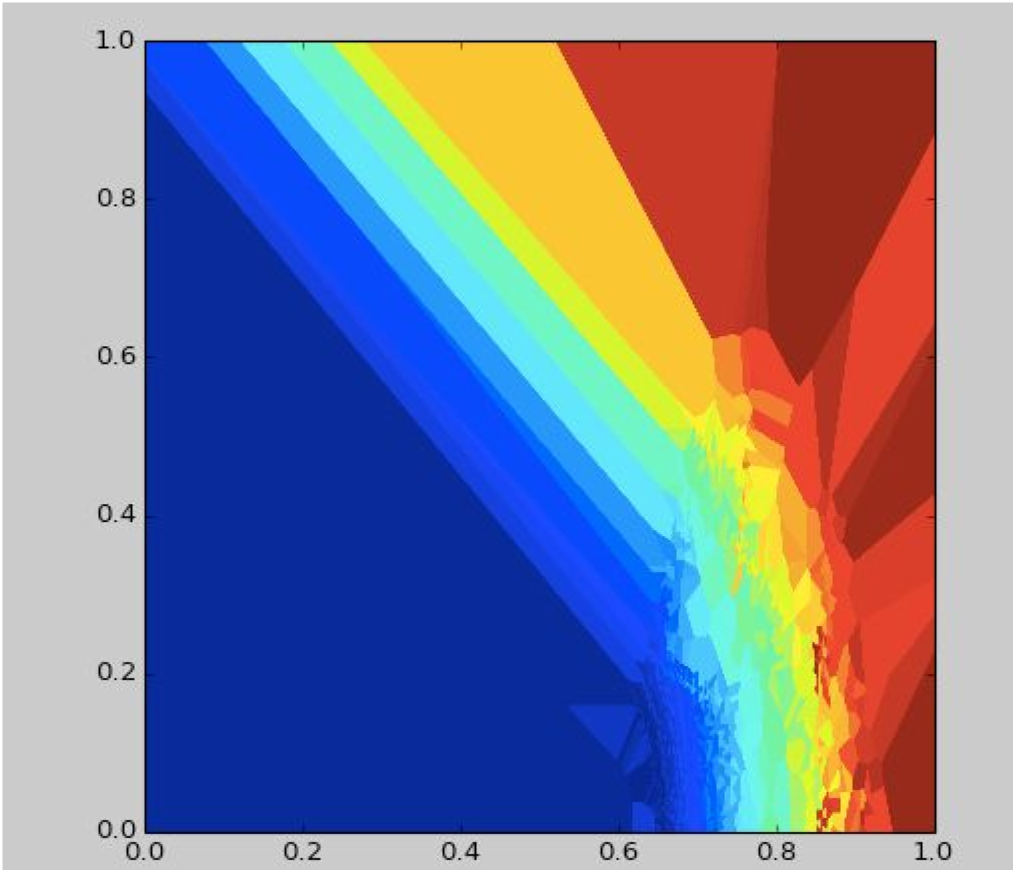
Result of the nearest:



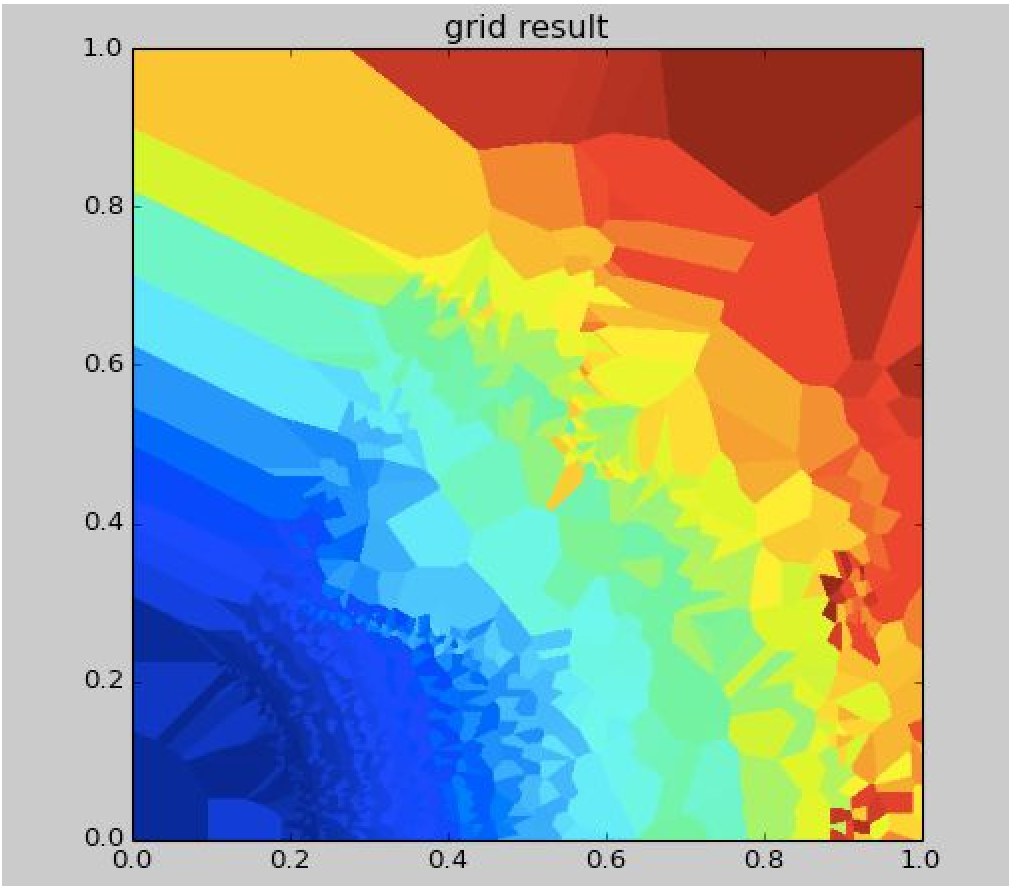
Result of the cubic:



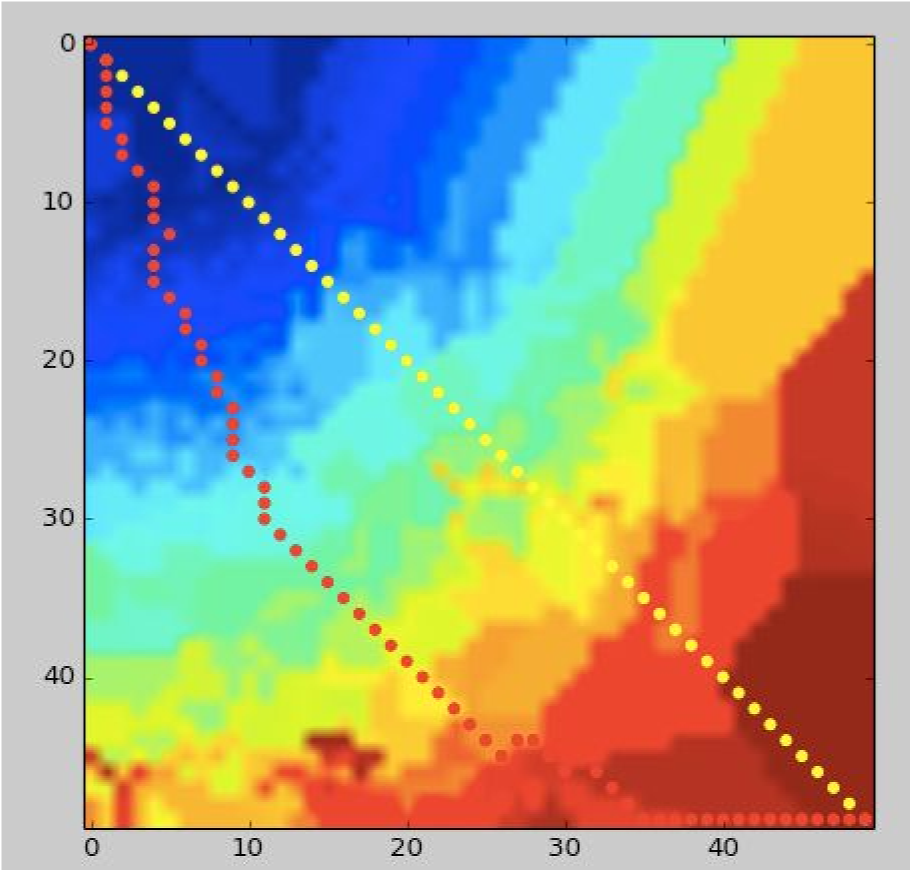
What is this mean?



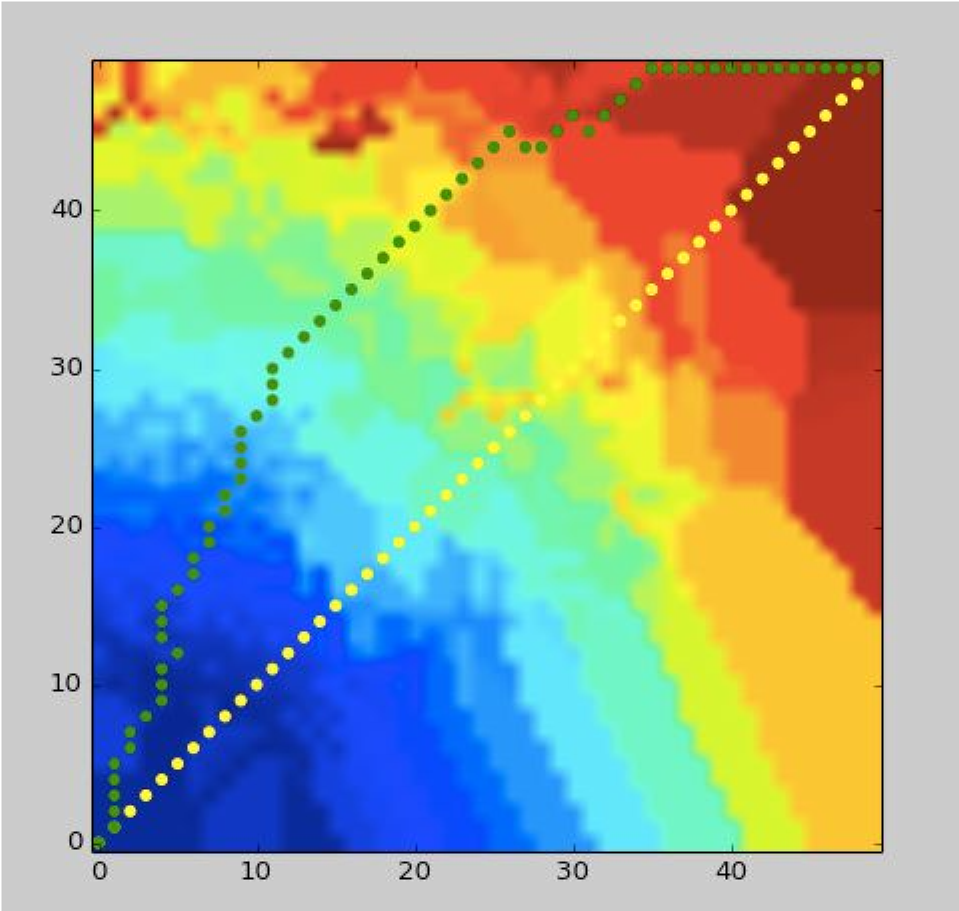
This (500 to 500) one is the one i am running the algorithm on



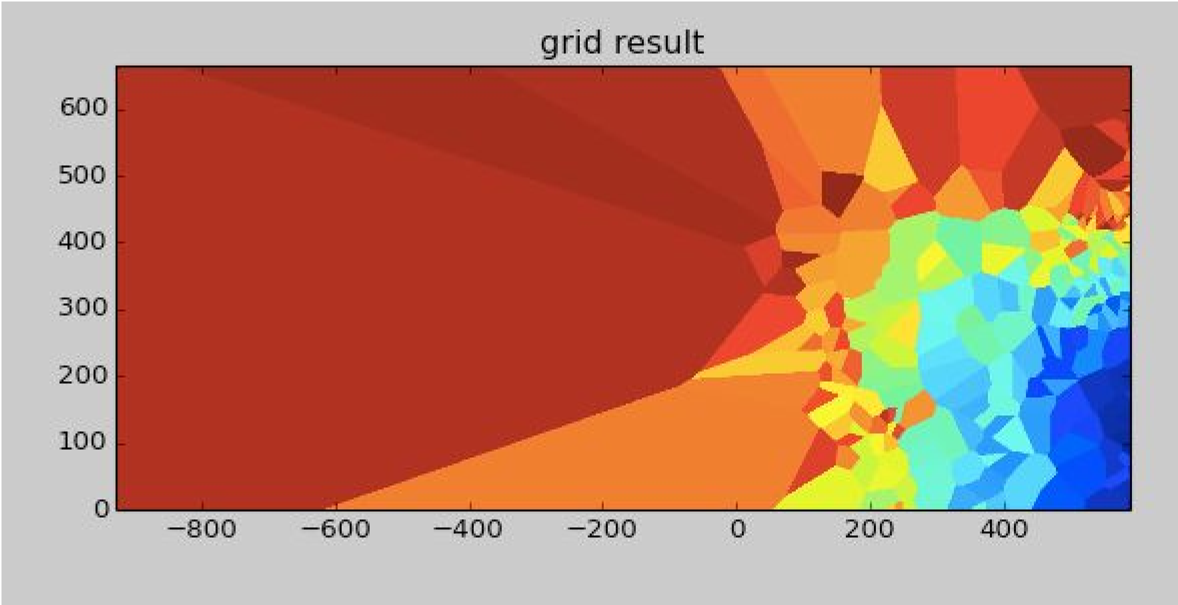
So it all worked, but i think plotting is off



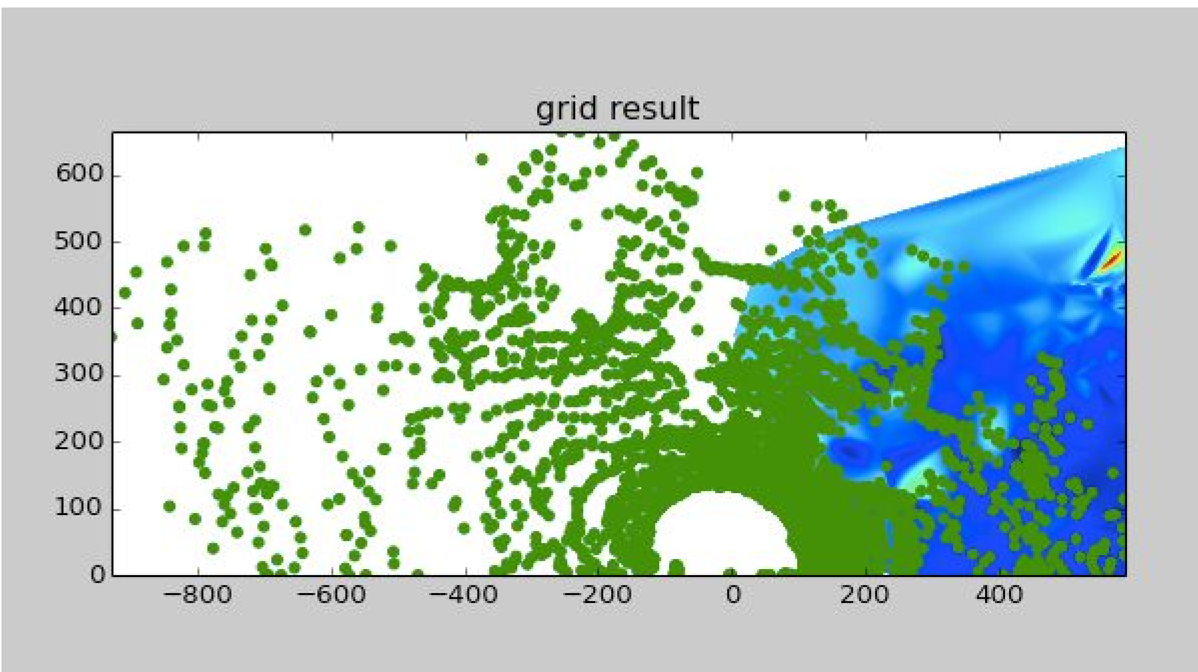
Even a better plot:



So, when we ran the test_3b code, the grid have this, which is the file we feed into the algorithm.



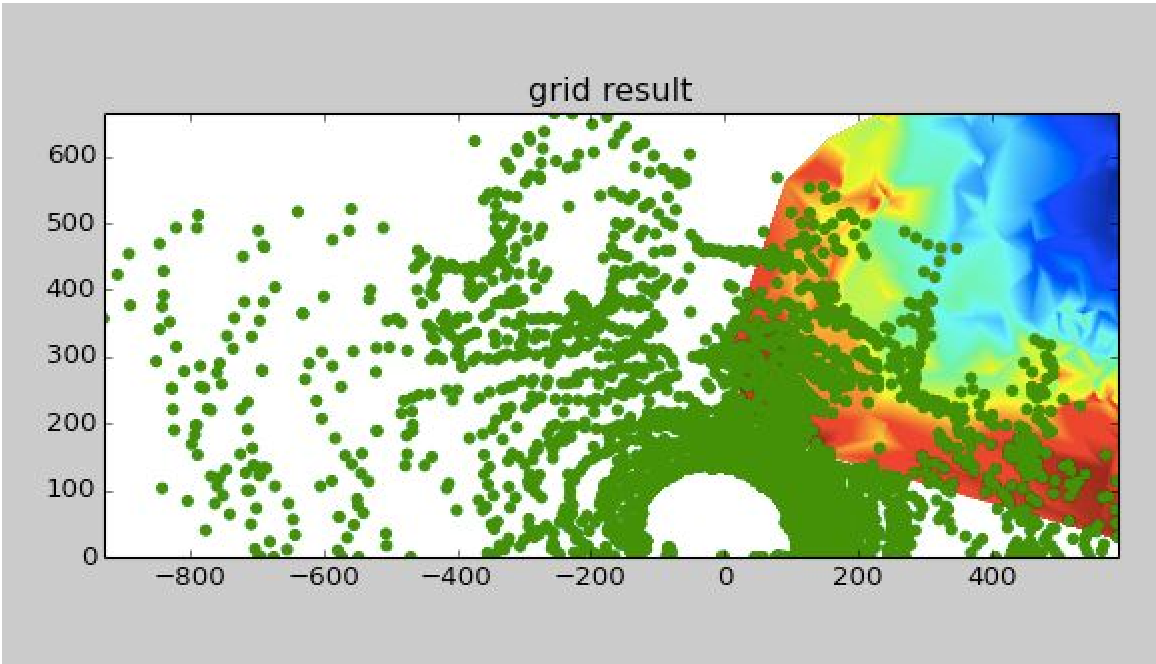
When i plotted against the point clous, something is clearly off :/ this is cubic



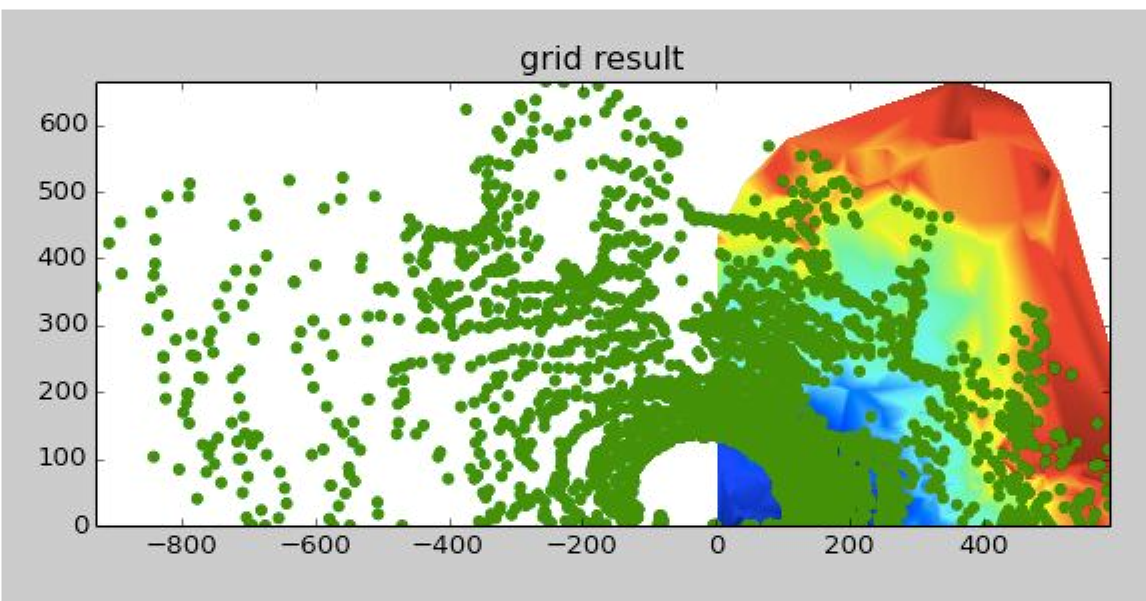

```
-930
588
0
665
```

Our x,y min and max values are -930 588 0 665 as printed, so the axes are in reasonable range…

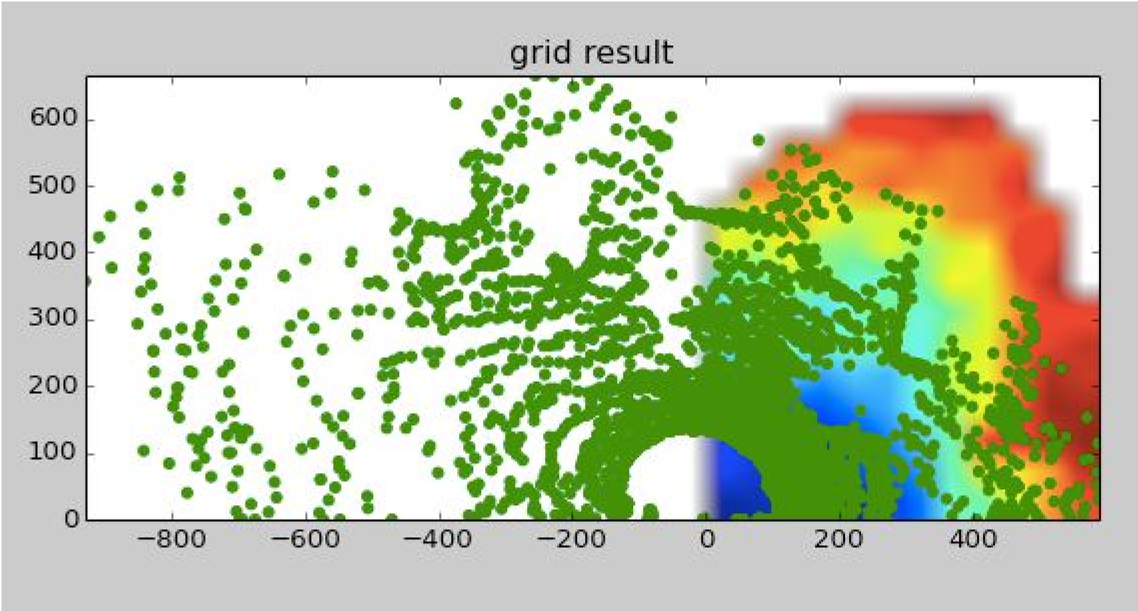
This is linear:



Then i tilted it, i think it makes more sense this way:



Then i changed the stepping from 1 to 50, so we have pixely images now



1. Avg slope
2. The whole area -- this involves
3. Do we follow the contour?
4. Change interpolation to rover size

How can we average the slope?

Presentation Outline

2-5 slides

1. the problem you were attempting to solve (including a brief overview of any concept(s) required for understanding the problem).

The aim of the my project was to employ the rover with self-driving capabilities.

Dynamic programming
D-star algorithm = works by taking the min-cost path from start to end point
Cost matrix = assigns a number to every point depending on its distance from

Terrain generating
DEM - digital elevation model
BFS
Advanced BFS - a second time around to average things
Smooth path with small degrees rover can rotate on.

2. a description of the solution(s) you pursued (with a show-and-tell of results, or show-and-tell of working hardware or partially-working hardware)

D-star algorithm.
Thinking in terms of making integration with other parts simpler:

1. **LIDAR to DEM happens on board**
2. **We only send DEM back, which is a simple text file**
3. **D-star executes on board (Dynamic programming takes O(n^2) time, whereas a naive approach would take exponential time to compute)**
4.

Place the good screenshots here.

3. a brief discussion of sources of error and/or technical challenges or setbacks encountered

Bugs.
I started off with modifying someone elses project I found on Github, but that didn't work so I wrote everything from scratch.
Implementing a dynamic programming table is tricky, I spent a good amount of time on this. I had to talk to 2 different CS professors to really come up with a way that will work.
Not documenting well enough - I had to go back and change things a lot.
Generating a DEM from LIDAR involved many steps. (Indexing with float coordinates) = a problem because I didn't really think about the physical space much.

4. a brief discussion of how you might have approached things differently if you had to do it over again.

1084 lines later...
I dived right into coding, before giving much thought on the fact that this was going to be a physical object. If Wes didn't point out to many drawbacks, such as making the rover fit into its paths, as well as

I realize that some students had multiple roles; these should of course be reported on separately, and in tandem with other students who worked on the same problem, as appropriate. So, think of the "5-8 min" as an average per student.
Note: this is not highly formal, and should not involve a huge amount of prep (as mentioned, a couple of slides per student is fine). Think of this as a conversation about what you did, with a few visuals of prototypes or relevant plots, of important relevant concepts, and of any significant results to date. The goal is to cover all subsystems separately. Then we can discuss actual or hypothetical integration of those systems in the Q&A.

FRIDAY 5/11

I didn't really take many notes this week, but I worked mostly with tracing bugs here and there.

Also tried to plot the LIDAR data yesterday night..

Problem: Currently, we are just ignoring the negative x and y values.
To reduce that, we will

TUESDAY 5/8
So, I made a mistake of not naming my variables in dem_generator well.
So now, I am having problems with updating them.
I get bugs, and I can't trace them well.

Okay, so I had weird bugs, and I spent time fixing them.

Problems Regarding the size of the Field:

What is the size of the field? For the DEM I am generating:
>> dem_x_dim_size = nx * 0.3556 #rover dimension
>> dem_y_dim_size = ny * 0.3556 #rover dimension

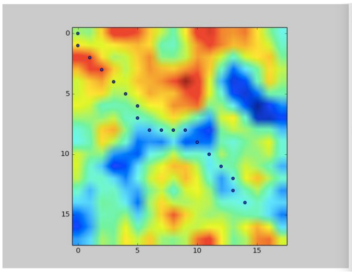
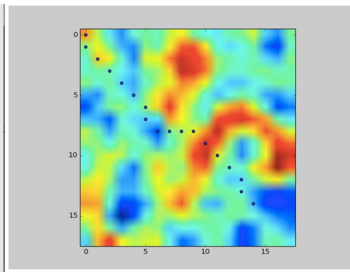
How do we measure it in LIDAR field?
>> We have points, and we crop out the size to fit into a certain dimension
>> Wes said 5 meters in each direction would be good enough.
5 / 0.3556 = 14.060742407 ⇒ let's say 14 rows and 14 columns

But, when we smoothen it, the shape becomes from (14,14) to (12,12)
Idea: I will try making dimensions (20,20) so that when we smoothen it, dimensions become (18,18) and then we crop out the far distances to make it (14,14)

Some problems involved deciding how to crop down the field. This might be an overkill, but I decided to just crop out the cols and rows beyond 14th, i.e. 5 meters from where we are, assuming that each cell is the size of the rover.

Questions raised, and then solved: Is it the distance from where we are to the farthest possible range?Should we just crop down the points that are far in terms of range in the sph.txt, and form a second sph? When to start binding? How to bind?

Problem: So I am having an issue with the path_finder. Somehow, the path is not being updated, and I am getting the same path regardless of the DEM. See the screenshot below. Feel free to expand it to see better.

Solved: It turns out that I am calling the dstar.py on an old dem that is not being updated, hence the same path is being generated all the time.

I reattempted at doing cost3 function work, but no it doesnt' produce a smooth path. Anyways, here is my code in main function from todays tryouts.

```
'''
#cost3 = np.array(cost2)
cost3 = np.empty(dem.shape)
cost3.fill(-1)
fill_cost3(dem, cost1, cost2, cost3, start_cell, goal_cell)
print "\nCOST3 MATRIX"
print DataFrame(cost3)

next_move = [] #initially empty list
output_path(cost3, start_cell, goal_cell, next_move, max_rotation_angle)
print "\nFrom " + str(start_cell) + " to " + str(goal_cell) + " the path is: "
print next_move

list_of_commands = path(initial_angle, size_of_cell, next_move) #returns instructions
print "Moving Instructions are "
print str(list_of_commands)
'''
```

FRIDAY 5/4 - MONDAY 5/7

Considerations

I haven't considered the size of the fields yet.
Initially smaller size pixels → so we have a nice high resolution image
Then downsample it
I.e. bind them
Max slope to do is about 10 degrees

Software Execution

LIDAR sph data comes in
(don't bind this)

We turn sph into cart
This code is on DEM_generator.py

Cart into DEM_original
I dont think we have this one yet
Make each cell contain one point only
Then fill nan values with averages

Create DEM_modified
Constraints are:
- make each cell 0.3556m x 0.3556 m

We send **DEM_original and DEM_modified and variables.py** to ground station
Ground station decides on a goal point (on the original or modified one???)
* Decide on a point on the modified one*

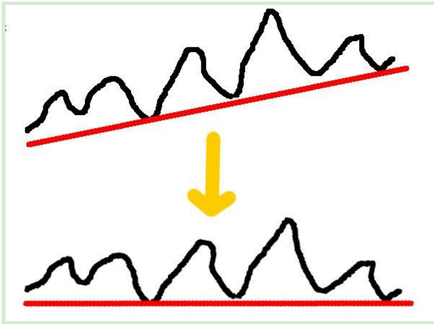
Ground station sends back **variables.py**
With updated goal_point,
We run the **elifs_dstar.py** algorithm

This generates the **instructions.txt**

Resolved: I am writing a bash script to execute these in order. However, dem_generator shows a DEM (python plot) that needs to be exited out for the rest of the code to function.

Problem: max_slope is not working.
Resolved: It turns out that the slope differences are very tiny on the DEM I am working with. So changing the max_slope value to something very small 0.005 helped! :)

Problem: To get a planar surface, **subtract a plane and fit** into topography, for situations like the one in the picture.



I found the code to help me, but apparently I cannot just use ArcPy without ArcGIS :/

```
import arcpy
from arcpy.sa import Trend, Raster
dem = 'c:/data/elevation.tif'
demmin = arcpy.GetRasterProperties_management(dem, "MINIMUM").getOutput(0)
dem = Raster(dem)
result = dem - Trend(dem) + demmin
result.save('c:/data/detrend.tif')
```

Wes' notes:

First you need to fit a plane to your surface. This gives you several parameters which can be used to define the plane. Then you can use your x and y arrays to compute a "z" array using the equation for the plane. Then subtract this new z array from your height map. No need to download ArcGIS. I can give you a plane-fitting method and library tomorrow. For now, I would read about the equation of a plane -- this will hopefully make things more clear.

TUESDAY 5/1

A problem: The current terrain I am working on is not sinusoidal, so probably I should try to use Wes' code to get better approximations.
I don't have the time to solve this so, this plan is aborted.

Done: I gave variables in variables.py more descriptive names and some commenting.
Do : Make sure that binding works on all maps
Do : Return an instruction list from d*

Executive decision: Successful trip is defined as reaching one point from initial point.

CROPPING THE SIZE of FIELD
Filtering = anything other than 5 meters go away
This will solve the problem I previously defined as

Falling off the cliff: At least, don't make big turns on edges, since this might result in falling off the edge of cliff.

Because now, the area is defined as small as we would like...

Size of the rover is 14x14 inches

MONDAY 4/30

Group MEETING NOTES

1. Sending text file with Radio Transmission - email them?
2. Jocelyn needs help with Arduino coding. I have never used it, but i took a few classes with C/C++ coding so I can hop into help with it.
3. Make turning so it involves curves

Arduino Coding

We need to have a serial monitor displaying data we are getting from LIDAR, angles from rotator

Cont rotation servo

Problem: convert rotation into time to theta, use that to generate a txt file

Lidar 270 measurement per second

ARDUINO:

how to export serial monitor to txt file >> Use putty

https://forum.arduino.cc/index.php?topic=367920.0

Get 2 types of data loaded into serial monitor (data from LIDAR and servo)

Speed to time to theta

Given angle put it into text file

data from arduino to pi

PROBLEM: Currently, the rover has trouble making in-place rotations, so I am working on smoothening the path. (For example. instead of having the algorithm day "Make a 90 degree turn", I would like to have it make a circle with a certain radius)

SOLVED:

- What is the maximum angle we want to turn by? Suppose it is 10 degrees.
- Previous command was "Turn by 45 degrees + Move 100 units"
- New command: "Turn 10 degrees + Move unit" *+ "Turn 10 degrees + Move unit" *+ "Turn 10 degrees + Move unit" * + "Turn 10 degrees + Move unit" *+ + "Turn 5 degrees"
    - Here 10 was given,
    - Units to move by was found by 100 * (10/45 = 0.22) = 22

- The previous path generator was **output_path** function. Editing this seems difficult, so I am going to write a new function to generate a smooth_path.

This all worked, here is the code and an output displaying execution :)

```
def smoothen_path(moving_instructions, max_angle):
	smooth_path = []
	for i, command in enumerate(moving_instructions):
		#if we are starting by moving forward, then just add this to list
		if (i==0 and type(command) is not tuple):
			smooth_path.append(command)
		if type(command) is tuple: #rotation case
			angle = command[0]
			rotation = command[1]
			if (angle <= max_angle): #we will NOT do slicing
				smooth_path.append(command)
			else:
				#Do slicing
				#Access the number to move forward by
				forward = moving_instructions[i+1] #this i+1 shouldn't be a problem since the last thing is always a float to move by
				num_times_to_loop = angle/max_angle
				for i in range(int(num_times_to_loop)):
					smooth_path.append((max_angle, rotation))
					smooth_path.append(forward * float(max_angle)/angle)
				#do the remaining angle and movement
				remaining_angle = angle - (int(num_times_to_loop) * max_angle)
				remaining_distance = forward - (forward * float(max_angle)/angle) * int(num_times_to_loop)
				if (remaining_angle > 0):
					smooth_path.append((remaining_angle, rotation))
					smooth_path.append(remaining_distance)

	return smooth_path
```

```
From (0, 0) to (0, 10) the path is:
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (0, 10)]
Moving Instructions are
[(90.0, 'R'), 10.0]

From (0, 0) to (0, 10) the SMOOTHENED path is:
[(10, 'R'), 1.1111111111111112, (10, 'R'), 1.1111111111111112, (10, 'R'), 1.1111111111111112, (10, 'R'), 1.11111111111
11112, (10, 'R'), 1.1111111111111112, (10, 'R'), 1.1111111111111112, (10, 'R'), 1.1111111111111112, (10, 'R'), 1.11111
11111111112, (10, 'R'), 1.1111111111111112]
```

IMPORTANT CONCERN: With this code, i turned everything into a smoothened path. What if there is an obstacle we are trying to avoid that now we run into due to smoothening???

SUNDAY 4/29

I did some HOUSEKEEPING WORK creating "variables.py"
Problem: I should probably create a separate file with information that needs to be transferred. This was we can radio communicate this file only.
Solution: I import this file on both functions I made " import variables as v". Here we can play with variables without touching the main algorithms.
TODO: Make the variable names more understandable, currently the names are not so descriptive.

SATURDAY 4/28

Finished converting alt/az/range to xyz stuff now! Details below.

Floating point indices = let's say each cell is 10 by 20 centimeters. Then, on a 3x5 grid, coordinates will map as follows:

X: 0-10 Y: 0-20	X: 0-10 Y: 20-40	X: 0-10 Y: 40-60	X: 0-10 Y: 60-80	X: 0-10 Y: 80-100
X: 10-20 Y: 0-20	X: 10-20 Y: 20-40	X: 10-20 Y: 40-60	X: 10-20 Y: 60-80	X: 10-20 Y: 80-100
X: 20-30 Y: 0-20	X: 20-30 Y: 20-40	X: 20-30 Y: 40-60	X: 20-30 Y: 60-80	X: 20-30 Y: 80-100

Good news: we don't need to int the floats anymore. I.e. if x is a float, we don't need to int(x)
How can we index then?
Do we know the number of rows and number of columns? No…
Maybe we don't need to know it actually..

We can just proceed with the information of sizes.

So, let's call **m = length of rows, n = length of cols, so each cell is mxn in size.**
Next, given point
```
2.020667218593133100e-15 3.300000000000000711e+01
4.370174432486667593e-01
```

We know the x coordinate maps to 0, y maps to 33, so this would be placed in row 0, column 33/20=1.

Indexing Math:
If we have range 0-29 in x coordinates and we need a 0-14 indexing integers, then we want
0,1 to map to 0
2,3 to map to 1
4,5 to map to 2
28,29 to map to 14 (so x/2, and 2 = (29+1)/(14+1))

If we are indexing to 0-9 only, then
0,1,2 to map to 0
3,4,5 to map to 1
6,7,8 to map to 2
27,28,29 to map to 9 (so x/3 and 3 = (29+1)/(9+1))
In an uneven case, if we are indexing to 0-3 only, then (8 = 30/4) **add 1 if 30%4 !=0
0 mapped by 0-7
1 8-15
2 16-23
3 24-29

REDO math: we have mxn and x_sizes x y_sizes given to us only.

The current cartesian coordinates for x range from 0 to 29, y range from 0 to 33…
So an idea is to map this to 29x33 grid for now, where each cell is 1x1.

THIS WORKED - and I can find a path on this DEM as well WOOHOO

Problem: The current path is just a straight line. This might be because the terrain is so flat, so its easier to move straight.
Solved: I changed the penalties we give to slopes by 10 folds.
Now, on the same terrain, we get a different path. See the screenshots below for before/after comparisons.
BEFORE (ascending slope factor 20)

```
From (0, 0) to (29, 33) the path is:
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (9, 9), (10, 10), (11, 11), (12, 12), (13, 13), (14, 14), (15, 15), (16, 16), (17, 17), (18, 18), (19, 19)
, (20, 20), (21, 21), (22, 22), (23, 23), (24, 24), (25, 25), (26, 26), (27, 27), (28, 28), (29, 29), (29, 30), (29, 31), (29, 32), (29, 33)]
Moving Instructions are
[(135.0, 'L'), 41.012193308819754, (45.0, 'R'), 4.0]
```

AFTER (ascending slope factor 200)

```
From (0, 0) to (29, 33) the path is:
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9), (9, 10), (9, 11), (10, 12), (11, 13), (12, 14), (13, 15), (14, 16), (15, 17), (16, 18), (17, 19),
(18, 20), (19, 21), (20, 22), (21, 23), (22, 24), (23, 25), (24, 26), (24, 27), (25, 28), (26, 29), (26, 30), (27, 31), (28, 32), (29, 33)]
Moving Instructions are
[(135.0, 'L'), 5.6568542494923806, (45.0, 'R'), 1.0, (45.0, 'L'), 7.0710678118654755, (45.0, 'R'), 1.0, (45.0, 'L'), 21.213203435596434, (45.0, 'R'), 1.0, (45.0, 'L'), 2.828427124
7461903, (45.0, 'R'), 1.0, (45.0, 'L'), 4.2426406871192857]
```

FRIDAY 4/27

Results of my conversation with Wes:

1. Don't int the floats, its oversimplification that will kill important information
2. Look up gridding with irregularly spaced data
3. A way to index into floating pointed array is
   `inds = np.nonzero((x==1.1) * (y==2.1))[0]`
   This works by creating an x matrix, where all cells indexing to row 1.1 is T, else F
   and a y matrix where all cells indexing to cols 2.1 is T, else F
   and multiplying them…
   So we can say `z[inds] = 10` for example.

Following the tutorial on here, I can actually fill in a matrix as follows. Notice the NaN values

	0	1	2	3	4	5	6
0	0.656497	NaN	0.601100	0.562957	0.665309	0.378666	0.331873
1	NaN	0.876075	0.728407	0.894970	0.716066	NaN	0.586653
2	0.983048	0.737942	0.664015	0.702500	0.977295	0.769767	0.864052
3	0.843781	NaN	NaN	0.692928	0.772268	0.798533	0.576458
4	0.687197	NaN	0.742811	NaN	0.569403	0.447941	0.402388
5	0.677722	NaN	0.627561	0.617612	0.833538	NaN	0.559407
6	0.828639	NaN	0.903514	0.920095	NaN	0.701597	NaN

It's probably the best to fill NaN values with averages around them… We can use the 8 surrounding cell averaging method as in the path-finding matrix.

BUT, first, I need to deal with the float indices problem.

So… there is a BaseMap library that I could potentially use, but I haven't looked into it yet.
Update: actually this is literally plotting points on maps so probably not that useful.
https://matplotlib.org/basemap/users/examples.html

WEDNESDAY 4/25 & THURSDAY 4/26

UPDATE WITH ALT/AZ/RANGE CALCULATIONS

Upon Wes' recommendation, we are no longer using the random code anymore…

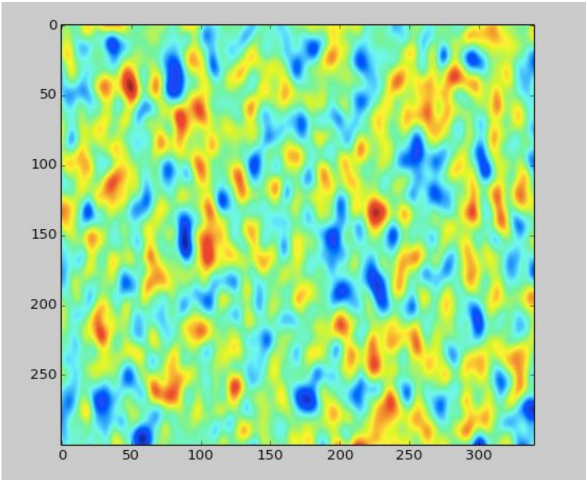
To do after getting familiar with his code:

- Convert DEM to ALT/AZ/Range
- Then treat this file as LIDAR input
- Real Challenge: convert this to DEM

Inspired by Wes' code, I wrote my own code to generate a DEM. Currently I am able to write it to a txt file.

```
import numpy as np
from pandas import DataFrame #to pretty print matrices
from scipy import signal

nx = 360
ny = 360
dem1 = np.random.rand(nx,ny)

sizex = 30
sizey = 10
x, y = np.mgrid[-sizex:sizex+1, -sizey:sizey+1]
g = np.exp(-0.333*(x**2/float(sizex)+y**2/float(sizey)))
filter = g/g.sum()

demSmooth = signal.convolve(dem1,filter,mode='valid')
# rescale so it lies between 0 and 1
demSmooth = (demSmooth - demSmooth.min())/(demSmooth.max() - demSmooth.min())

print "DEM1 SHAPE is " + str(dem1.shape)
print "DEMSMOOTH SHAPE is " + str(demSmooth.shape)

#print DataFrame(demSmooth)
np.savetxt("dem.txt", demSmooth)

import matplotlib.pylab as plt
plt.imshow(dem1)
plt.imshow(demSmooth)

plt.show()
```

This generates a DEM that can be visualized as the following:



Now, I converted this to ALT/AZ/Range, I call it the sph.txt

```
#now turn this into Spherical coordinates and save in sph.txt
sph = [] #np.empty(demSmooth.shape)
def cart2sph(x,y,z):
    XsqPlusYsq = x**2 + y**2
    r = np.sqrt(XsqPlusYsq + z**2)                # r
    elev = np.arctan2(z,np.sqrt(XsqPlusYsq))      # theta
    az = np.arctan2(y,x)                           # phi
    return (r, elev, az)

def cart2sphA(pts, sph):
    for x in range(pts.shape[0]):
        for y in range(pts.shape[1]):
            z = pts[x][y]
            #sph[x][y] = cart2sph(x,y,z)
            sph.append(cart2sph(x,y,z))

cart2sphA(demSmooth, sph)

np.savetxt("sph.txt", sph)
```

One problem with this is that, we have to put it into matrix format, now it's just a list of points… I suppose, this is the format in which its going to come…

I observed that, the points in the cart.txt are ranked by their x coordinates first. Also, the x coordinates decrease slowly… Here is an example that shows how end of a row, and the beginning of another one, separated by the 1-0-6 line as highlighted below.

```
1.775737858763662212e-15 2.900000000000000000e+01 7.991124155139479601e-01
1.836970198721029983e-15 3.000000000000000355e+01 8.320791871997316180e-01
1.898202538678397557e-15 3.100000000000000000e+01 8.106418306263502016e-01
1.959434878635765131e-15 3.200000000000000000e+01 6.890027903158567l8e-01
2.020667218593132706e-15 3.300000000000000000e+01 4.151836864292452467e-01
1.000000000000000000e+00 0.000000000000000000e+00 6.872255677344101255e-01
1.000000000000000222e+00 1.000000000000000000e+00 8.270902088532308127e-01
1.000000000000000222e+00 2.000000000000000000e+00 7.275571332110390976e-01
1.000000000000000000e+00 2.999999999999999556e+00 5.965786256658969222e-01
```

So, I found an amazing tutorial here to do this with indices as integers
http://chris35wills.github.io/gridding_data/

BUT our indices (x,y points) are not integers, they are floats… How can I make them integers so that i know how to plot them??

OBSERVATION: The biggest we get with e's are e+01 (1400 found) , the smallest is e-17 (2 found)

IDEA: I need to simplify these points. Turning them to integers might be oversimplification… But I will do that for now, so that I can at least get an approximate height map.

TUESDAY 4/24

Now, I am working on debugging path-finder.

- **Problem: Output_path** is sometimes buggy == Test it on many test cases

So, testing on a case revealed that the cost3 function doesn't work properly.
Here are screenshots of the elevation matrix, cost2, cost3.
Clearly, we want to avoid the cell [2,3] since it has an elevation of 100, i.e. its an obstacle or a big hill. Cost2 avoids it (value at 2,3 is 10022), cost3 doesn't see it (value is 18).
This demonstrates that, the problem is with cost3 function, not the output_path function.

ELEVATION MATRIX

	0	1	2	3
0	0.0001	-0.0299	-0.1100	0.0000
1	-0.0088	1.8559	-0.2729	0.0130
2	-0.0137	0.2289	2.4338	100.0125
3	0.0000	0.1099	0.1107	0.0000

COST2 MATRIX

	0	1	2	3
0	47.215143	44.335600	41.314729	43.222714
1	38.534700	30.564000	31.151829	33.196714
2	34.288871	24.195129	15.738429	100022.500000
3	30.332100	20.112300	10.110700	0.000000

COST3 MATRIX

	0	1	2	3
0	37.157879	37.626959	35.382602	34.645040
1	34.573269	35.311978	31.690375	30.930016
2	30.966663	28.480656	20.921276	18.039534
3	28.527543	26.135310	16.653648	0.000000

FIRST ATTEMPT AT SOLVING THIS

One problem was that, when averaging with fill_cell_with_average function, we weren't adding the value of the current cell, thus the **cost3 matrix** wasn't filled up correctly.
Once I changed it so that the value of the cell from cost2 also goes into average calculation, this is what I got:

COST3 MATRIX

	0	1	2	3
0	328.582014	721.358029	1258.588912	1408.640080
1	420.946214	507.220185	2432.253635	3117.769242
2	480.156010	895.693486	1872.279857	16685.449612
3	599.659200	992.455202	3102.024600	0.000000

THIS GAVE RISE TO ANOTHER PROBLEM

So now, clearly, we cannot find a path from [0,1] to [3,3], since the values increase by a lot as we approach [3,3]. In particular, there is no path to follow from [1,1] to [3,3].

To solve this, I will not average on the all neighbors, I will average on only neighbors that are facing the goal_point, the way we did in fill_cost2 function. Here are the changes that will go into this:

- Cost3 will be initialized to a -1 matrix, instead of the copy of cost2.

This still doesn't solve the problem:

COST3 MATRIX

	0	1	2	3
0	40.704914	36.841539	35.889997	35.857086
1	36.968914	25.818086	25025.646743	33357.145048
2	29.539000	22.352017	33346.079476	50011.250000
3	27.232100	17.539139	25012.087282	0.000000

What happens at point [1,1] that makes the value much smaller? - I don't know yet :(
Update: turns out its not that important, i am using the other matrix

If I average values from the cost2 function, then the cells that are not neighboring aren't affected by the high slope.
Maybe if we were working on a much bigger matrix, this wouldn't be a problem.
So I leave cost3 as dependent on cost2 averages only.
I tried many cases where cost3 depended on its averaged values… This just creates dead ends…

Running the code on many cases with obstacles convinced me that COST2 function is actually reliable. So, I will assume we have it working for now. Here are screenshots of different DEM's that are working (make them bigger to see the obstacle arrangements I used)

```
ELEVATION MATRIX
        0       1         2       3
0  0.0001 -0.0299   -0.1100  0.0000
1 -0.0088  1.8559   -0.2729  0.0130
2 -0.0137  0.2289  100.0125  2.4338
3  0.0000  0.1099   90.1107  0.0000

COST1 MATRIX
      0     1     2     3
0  42.0  38.0  34.0  30.0
1  38.0  28.0  24.0  20.0
2  34.0  24.0  14.0  10.0
3  30.0  20.0  10.0   0.0

From (0, 0) to (3, 3) the path is:
[(0, 0), (1, 1), (2, 2), (3, 3)]
Moving Instructions are
[(135.0, 'L'), 4.242640687119285 7]

COST2 MATRIX
           0          1             2        3
0  54.504086  44.474086     40.463414  37.3014
1  56.158714  42.429314     30.300514  27.2754
2  55.144143  44.658943  71451.500000  12.4338
3  58.985943  54.896943  62652.790943   0.0000
```

```
ELEVATION MATRIX
        0       1         2       3
0  0.0001 -0.0299   -0.1100  0.0000
1 -0.0088  1.8559   -0.2729  0.0130
2 -0.0137  0.2289  100.0125  2.4338
3  0.0000  0.1099    0.1107  0.0000

COST1 MATRIX
      0     1     2     3
0  42.0  38.0  34.0  30.0
1  38.0  28.0  24.0  20.0
2  34.0  24.0  14.0  10.0
3  30.0  20.0  10.0   0.0

From (0, 0) to (3, 3) the path is:
[(0, 0), (1, 1), (2, 2), (3, 3)]
Moving Instructions are
[(135.0, 'L'), 4.2426406871192857]

COST2 MATRIX
           0          1             2        3
0  54.504086  44.474086     40.463414  37.3014
1  38.534700  42.429314     30.300514  27.2754
2  34.288871  24.195129  71451.500000  12.4338
3  30.332100  20.112300     10.110700   0.0000

From (0, 0) to (3, 3) the path is:
[(0, 0), (1, 0), (2, 1), (3, 2), (3, 3)]
```

```
ELEVATION MATRIX
        0        1         2       3
0  0.0001  -0.0299   -0.1100  0.0000
1 -0.0088  80.8559   -0.2729  0.0130
2 -0.0137   0.2289  100.0125  2.4338
3  0.0000   0.1099   90.1107  0.0000

COST1 MATRIX
      0     1     2     3
0  42.0  38.0  34.0  30.0
1  38.0  28.0  24.0  20.0
2  34.0  24.0  14.0  10.0
3  30.0  20.0  10.0   0.0

From (0, 0) to (3, 3) the path is:
[(0, 0), (1, 1), (2, 2), (3, 3)]
Moving Instructions are
[(135.0, 'L'), 4.2426406871192857]

COST2 MATRIX
           0             1             2        3
0  54.504086     44.474086     40.463414  37.3014
1  58.489157  71492.866571     30.300514  27.2754
2  55.144143     44.658943  71451.500000  12.4338
3  58.985943     54.896943  62652.790943   0.0000

From (0, 0) to (3, 3) the path is:
[(0, 0), (0, 1), (1, 2), (2, 3), (3, 3)]
```

- ★ **Updating the cost function:** Should we make it dependent on the degree to turn?
  - We can probably do that. Here is a way I am thinking about it…
  - We create cost2, find a path, then we get instructions.
    - If we find a degree of turning that's greater than what we would like, then we find another path.
    - We check the surroundings for obstacles.
    - We make a slower turn…

- ★ Is it that important to have a slower angle turn at everywhere?
- ★ We need to check for corner/edge cases since this might create a chance of falling off the cliff.

BINDING a 2D matrix to smaller sizes

Let's call the matrix size mxn, where m is num_rows, n is num_cols

Now, let's call the rover size axb.

Do we want to make each rows length a and each cols length b or vice versa? Does it matter?

Since the rover is going to turn and move in all directions, we

---

MONDAY 4/23

- **Code to generate random data:** The function gives a pt-cloud data where every point is specified in range, alt, az, mimicking what LIDAR would output.
- **Assumptions**: alt (-90, 90) and az (0, 360) and range (0, 100cm)

```
def generate_terrain(file, num_rows):
    for r in range(num_rows):
```

```
        file.write(str(randint(-90,90))) #alt btw 90,-90
        file.write(" ")
        file.write(str(randint(0,360))) #az btw 0,360
        file.write(" ")
        file.write(str(randint(0,1000))) #range in cms
        file.write(" ")
        file.write("\n")
```

- **Objective**: Given a point cloud, make a DEM. Origin is 0,0,0. Assume size of each grid is known.
- **Should we sort the data?** It's random now, should it be sorted by alt or az or distance? >> If so, we can generate the data in some other way.
- **How to convert this to DEM?**
  - Las2dem is a tool, I can try to figure out how to use it: http://www.cs.unc.edu/~isenburg/lastools/download/las2dem_README.txt

I'm looking at Jocelyn's notes to understand lidar data: (I cannot really undertand how the following code is going to work, so I will ask about that in class tomorrow.)

The collected data will be in terms of the altitude angle phi Φ, the azimuth angle theta θ, and the distance measured on the LIDAR sensor D. In order to record these data, the mount will move in the following way: the altitude servo motor, hereby referred to as the ALT motor, will start at some starting altitude angle Φ_0; the azimuth servo motor, hereby referred to as the AZ motor, will start at some starting azimuth angle θ_0. The AZ motor will then complete one full rotation at a rate defined by the user, returning to θ_0. As this is happening, the ALT motor holds its position at Φ_0. When the AZ motor returns to its starting position θ_0, the ALT motor will shift the LIDAR scanner to a new position Φ (defined by the user), and the whole process happens again for this new Φ. Figure 2 shows a sketch of this process and the point cloud product. Each point measurement is recorded as a spherical coordinate point (Φ, θ, D) using two `for` loops, presented in the simple following example:

```
phi = [some array]
theta = [some array]
for altitude in phi:
      for azimuth in theta:
            r = measure_range()
            ranges[azimuth, altitude] = r
            step_theta()
      step_phi()
```

These points are then converted into an (x,y,z) point using trigonometric calculations outlined in Figure 3. These xyz points are measured relative to the LIDAR scanner located at the origin. As the rover moves, the origin changes, meaning that the xyz points will also change. In order to account for these differences, we will calculate the total distance between the two origins by trigonometrically calculating it relative to a reference point on the outcrop that appears on two different LIDAR scans (ex. a tree or an otherwise distinctive point). These xyz data will be transmitted over radio to “Mission Control,” where we can then plot these points onto a point cloud with a program like SlugView[1]. Once these data are plotted in a point cloud, we will combine these points with photos taken from cameras mounted on the mast to create a 3D photo construction that is “human-interfaceable,” meaning that we can use this 3D construction to see and understand points of interest that are in the rover’s field of view. Once we identify a point of interest in the construction, we can command ScOut to move to an xyz point within that construction. The rover will then move to that point with the guidance of our Machine Vision algorithm. This process repeats for every point of interest that the rover moves to.

Finally, the scanning time depends on the size of $\Delta\theta$ and $\Delta\Phi$ and the number of scans per second. The LIDAR Lite v3 can record 270 scans per second. Table 1 shows the total scanning times of this LIDAR scanner at different levels of precision (ie. number of measurements taken per rotation) assuming that Φ covers a 45° range (this number might increase or decrease depending on the rover’s needs).

FRIDAY 4/20

DONE: Meba started to write the code for moving the motors, I am putting her code inside the for-loops and while-loops I am writing.
DONE: Put a big penalty for intense slopes… Update this variable in the main function :)

PROBLEM: Sometimes the path function doesn’t work well, gets into infinite loop…
Ex: We had a problem finding the path on this matrix

COST3 MATRIX - note that the values are too big, bcs I was multiplying by 100.

	0	1	2	3
0	686.286881	886.737714	479.034286	0.000000
1	709.229312	822.951500	1597.168571	2321.383429
2	635.461684	781.199038	1869.055874	1396.304752
3	629.305955	879.256099	1127.694741	1464.351789

Goal: Identify the problem with **output_path function**, we can’t have it on when we are on the mission

First - I will try the code I found a few days ago, maybe that’s just better code… I call it the **output_path2 function - NOT WORKING**

PROCEDURAL LANDSCAPE GENERATION
Work on generating DEM from simulated data

---

[1] https://websites.pmc.ucsc.edu/~seisweb/SlugView/manual/manual.html

1. So I did download Terragen, which promised to output DEMs. Apparently the Terragen version 4 eliminated DEMs, since it is seen as a very old file type.
2. Then I looked at USGS. Entered Wellesley's coordinates to find the outcrop. Looks like no LIDAR data is available here, but there are other file types, which unfortunately I am not sure what they mean. This is the website I used https://earthexplorer.usgs.gov
3. I can find the coordinates of the outcrop on Google Maps, but I don't see it necessary just yet.
4. A list of terrain tools and software packages I looked at: http://vterrain.org/Packages/Artificial/
5. Artificial terrain generation: http://vterrain.org/Elevation/Artificial/
6. **Finally, this is a good one: http://www.playfuljs.com/realistic-terrain-in-130-lines/ but it's in JS and I need to translate it to Python to use…**
7. **There is the Diamond-square algorithm, but it might take a while to implement + get a DEM https://en.wikipedia.org/wiki/Diamond-square_algorithm**
8. **I found some LIDAR-derived DEM's but I dont have a way of opening them… Downloaded here https://nationalmap.gov/3DEP/3dep_prodmetadata.html More information on ArcGIS also here http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=Best_practices_for_building_terrain_datasets**

Name	Date Modified	Size	Kind
FESM_OPR_PROJ.cpg	Apr 17, 2018 at 6:05 PM	5 bytes	Documen
FESM_OPR_PROJ.dbf	Apr 18, 2018 at 11:38 PM	944 KB	Documen
FESM_OPR_PROJ.prj	Apr 17, 2018 at 6:05 PM	167 bytes	Documen
FESM_OPR_PROJ.shp	Apr 18, 2018 at 11:38 PM	45.8 MB	Documen
FESM_OPR_PROJ.shx	Apr 18, 2018 at 11:38 PM	7 KB	Documen
FESM_OPR_TILE.cpg	Apr 17, 2018 at 6:05 PM	5 bytes	Documen
FESM_OPR_TILE.dbf	Apr 18, 2018 at 11:38 PM	624.6 MB	Documen
FESM_OPR_TILE.prj	Apr 17, 2018 at 6:05 PM	167 bytes	Documen
FESM_OPR_TILE.shp	Apr 18, 2018 at 11:38 PM	78 MB	Documen
FESM_OPR_TILE.shx	Apr 18, 2018 at 11:38 PM	4.6 MB	Documen

THURSDAY 4/19

NOTES FROM MEETING W CHRISTINE
The best working idea seems to be this:

1. Finish the cost2 matrix using BFS starting from the goal node
2. Implement a cost3 matrix
    a. This will follow the same traversal as the cost2, except that it will average the surrounding values
        i. CONCERN: Should we avg all 8 surrounding values or should we just avg ones facing the goal point?
3.

Smart BFS

1. Copy the Array
2. Number is starting on top left corner with 0
3. Now, each number can be accessed by coordinates [num

Plan until the end of the week

- ❏ DONE- Implement BFS in fill_cost2 function

1. Check the starting node and add its neighbours to the queue.
2. Mark the starting node as explored.
3. Get the first node from the queue / remove it from the queue
4. Check if node has already been visited.
5. If not, go through the neighbours of the node.
6. Add the neighbour nodes to the queue.
7. Mark the node as explored.
8. Loop through steps 3 to 7 until the queue is empty.

FIXED PROBLEM - So I implemented but the cost2 matrix doesn't end up nicely, I need to fix it somehow… Is it the way we implement it? There are a few same values that I think gives the matrix complications… :(:(:(

See the code on github for updates…

  - ❏ Generate a path2 - **this still requires fixing, problem when we**
- ❏ DONE Implement BFS in fill_cost3 function
  - ❏ Generate a path3
- ❏ Compare path2 and path3
- ❏ Email Christine the results

The paths seem to be generated kinda randomly, I need to understand them.

IMPORTANT NOTE ABOUT PATH TRACER
- We currently assume that we are facing RHS
- We don't have a way of changing this yet
- This results in some large angles, like 135 degrees Left etc.
- Here is an example

```
COST2 MATRIX
          0          1          2         3
0  30.330100  20.300100  10.220000   0.0000
1  34.315171  25.624214  14.389857  10.0130
2  39.233486  28.748286  25.742143  30.0125
3  43.075286  38.986286  40.388343  43.2190
```

From (3, 3) to (0, 3) the path is:
[(3, 3), (2, 2), (1, 3), (0, 3)]

Instructions are [(135.0, 'L'), 1.4142135623730951, (90.0, 'R'), 1.4142135623730951, (45.0, 'L'), 1.0]

UPDATE - Fixed the issue, here is the same matrix…

Fixed

```
COST2 MATRIX
          0          1          2         3
0  30.330100  20.300100  10.220000   0.0000
1  34.315171  25.624214  14.389857  10.0130
2  39.233486  28.748286  25.742143  30.0125
3  43.075286  38.986286  40.388343  43.2190

From (3, 3) to (0, 3) the path is:
[(3, 3), (2, 2), (1, 3), (0, 3)]
Moving Instructions are
[(45.0, 'L'), 1.4142135623730951, (90.0, 'R'), 1.4142135623730951, (45.0, 'L'), 1.0]
```

TUESDAY 4/17

I decided to follow a dynamic programming tutorial to get done with this implementation, which is taking longer than I thought, probably I am doing something very wrong.

Link to tutorial https://web.stanford.edu/class/cs97si/04-dynamic-programming.pdf

The 3 step approach:

1. Define subproblem
2. Recurrence
3. Solve base cases

Subproblem: -- this is the implementation of `cost2` function
Each cell is surrounded by 8 other cells.
Compute value + elevation
Take the min of these values to fill the middle one

So the code to compute val from only one of the 8 cells is this

```
        # value at cell num 0 [x,y+1]
        a = x
        b = y+1
        temp_val = cost2[a,b]
        print cost2[a,b]
        distance = 10 if math.fabs(a+b-x-y) == 1 else 14
        print distance
        temp_val += distance
        slope = (dem[a,b] - dem[x,y])/ distance
        print slope
        temp_val += math.fabs(slope) * ascending_slope_factor if slope < 0 else
math.fabs(slope) * descending_slope_factor
        print temp_val
```

Now, I the code to compute the minimum val from surrounding 8 cells is

```
for i in range(8):
                if i == 0:
                        a = x
                        b = y+1
                elif i == 1:
                        a = x-1
                        b = y+1
                elif i == 2:
                        a = x-1
                        b = y
                elif i == 3:
                        a = x-1
                        b = y-1
                elif i == 4:
                        a = x
```

```
                b = y-1
            elif i == 5:
                a = x+1
                b = y-1
            elif i == 6:
                a = x+1
                b = y
            else:
                a = x+1
                b = y+1
            temp_val = cost2[a,b]
            distance = 10 if math.fabs(a+b-x-y) == 1 else 14
            temp_val += distance
            slope = (dem[a,b] - dem[x,y])/ distance
            temp_val += math.fabs(slope) * ascending_slope_factor if slope < 0 else
math.fabs(slope) * descending_slope_factor
            '''print i
            print cost2[a,b]
            print distance
            print slope
            print temp_val'''
            if temp_val < temp:
                temp = temp_val
                bt[x,y] = i
```

Recurrence or the problem of how to traverse:

We have been starting at the goal cell and going to the start cell i.e. backwards

But how we traverse changes the results

Should we do traverse more than once?

Next, the question is which loop we put this function in, i.e. where do we start from exploring?

Then do the 4-quadrant model as in the naive cost function ?

Then re-update the whole thing? Wouldn't this result in very high values instead of updating?

Problem definition: How we fill the matrix changes the solutions…

Because each value depends on the 8 surrounding values

We have copied the cost1 matrix into cost2, so we don't deal with -1s

But perhaps we should have started with -1s?

We can do it recursively: start at cell [0,0] and then each cell calls the other one…

What would be the base case?

If base case is target, we will return 0

There is also this tutorial
https://www.cs.cmu.edu/~motionplanning/lecture/AppH-astar-dstar_howie.pdf
Inspired by this, here is my attempt at writing BFS-like code

```
Start with 3 arrays
Unvisited = all of the 2D array, where the goal_cell is on top
Visiting
Visited

Now, pop from the unvisited i.e. pop the goal_cell
Queue it to visiting

Pop neighbors of the visiting
Queue them to visiting
Q goal_cell to visited
```

```
NOTES ON FILL_MATRIX
''' components to calculate from cost2[a,b] where a,b are max 1
unit away from x,y
          val_of_prev_cell = cost2[a,b] #this is assumed to be
filled before, if not what to do?
          distance = fabs(cost1[a,b] - cost1[x,y]) #might need to
change it with 10 or 14
          slope = (dem[a,b] - dem[x,y])/ distance
          if slope < 0 (ascending), use ascending_slope_factor
          else use descending_slope_factor
     ''
```

Base cases:
If there is only 1 cell on the whole map -- error in binding the grids
If we are at target cell, return 0

I can see the horizon: Once I implement it, I can use this code to find the min cost path
UPDATE: ended up implementing my own version and didn't use these, but just keeping here in case mine is buggy or something…

```
# A Naive recursive implementation of MCP(Minimum Cost Path) problem
R = 3
C = 3
import sys
```

```
# Returns cost of minimum cost path from (0,0) to (m, n) in mat[R][C]
def minCost(cost, m, n):
    if (n < 0 or m < 0):
        return sys.maxsize
    elif (m == 0 and n == 0):
        return cost[m][n]
    else:
        return cost[m][n] + min( minCost(cost, m-1, n-1),
                                minCost(cost, m-1, n),
                                minCost(cost, m, n-1) )

#A utility function that returns minimum of 3 integers */
def min(x, y, z):
    if (x < y):
        return x if (x < z) else z
    else:
        return y if (y < z) else z


# Driver program to test above functions
cost= [ [1, 2, 3],
        [4, 8, 2],
        [1, 5, 3] ]
print(minCost(cost, 2, 2))
```

```
# Dynamic Programming Python implementation of Min Cost Path
# problem
R = 3
C = 3

def minCost(cost, m, n):

    # Instead of following line, we can use int tc[m+1][n+1] or
    # dynamically allocate memoery to save space. The following
    # line is used to keep te program simple and make it working
    # on all compilers.
    tc = [[0 for x in range(C)] for x in range(R)]

    tc[0][0] = cost[0][0]

    # Initialize first column of total cost(tc) array
    for i in range(1, m+1):
        tc[i][0] = tc[i-1][0] + cost[i][0]

    # Initialize first row of tc array
    for j in range(1, n+1):
        tc[0][j] = tc[0][j-1] + cost[0][j]
```

```
    # Construct rest of the tc array
    for i in range(1, m+1):
        for j in range(1, n+1):
            tc[i][j] = min(tc[i-1][j-1], tc[i-1][j], tc[i][j-1]) + cost[i][j]

    return tc[m][n]

# Driver program to test above functions
cost = [[1, 2, 3],
        [4, 8, 2],
        [1, 5, 3]]
print(minCost(cost, 2, 2))
```

MarMon

Critical Design review is tomorrow
Here are my notes:

1. Major design changes

I was going to use the A* algorithm, but given the need to make up to exponential time calculations, I decided to use the D* algorithm instead.
It would be easier to use A* but the rover would then function much more slowly.
It is more difficult to implement D*, since it uses infamous dynamic programming, but it will give us the path we are looking for much faster.

2. Summary of progress

I implemented a D* algorithm that finds the shortest viable path the rover can traverse.
I also generated the list of commands that will move the motors (Turn 30 degrees left, go straight for 2 ft etc.)

3. Goals for next week

I. Finish the dynamic programming table implementation - this will make us have a path that avoids steep slopes.
II. Fine tune the algorithm: avoid paths that make the rover turn more than 45 degrees, avoid paths that are too narrow that they can cause the rover to get squeezed, avoid paths that have unseen obstacles (the thin tree that LIDAR might not detect but IR sensors will)

4. Major issues that I want comment on from the class as a whole

None.

Also here is a summary of benefits of D* based on my research:
https://www.cs.cmu.edu/~motionplanning/lecture/AppH-astar-dstar_howie.pdf
• *Stands for "Dynamic A* Search"*
• *Dynamic: Arc cost parameters can change during the problem solving process—replanning online*

• *Functionally equivalent to the A* replanner*
• *Initially plans using the Dijkstra's algorithm and allows intelligently caching intermediate data for speedy replanning*
• *Benefits – Optimal*
– Complete
– More efficient than A replanner in expansive and complex environments*
• *Localchangesintheworlddonotimpactonthepathmuch*
• *Mostcoststogoalremainthesame*
• *It avoids high computational costs of backtracking*

FRIDAY 4/13

FINISHED goal of the day: Return path of points as "go straight, turn left by x degrees" commands. Here is the worksheet I made to write bugless code.

```
Ex [(1, 0), (1, 1), (1, 2), (1, 3), (1, 4)]

>> Go straight from (1,0) to (1,4) i.e. 4 units

Ex [(1, 0), (2, 1), (3, 2), (3, 3), (3, 4)]

>> Start at (1,0)
We are facing the 0 radians direction

>> Next one is (2,1)
Face 3/2pi radians "Turn 45 degrees to Right"

>> Next one is (3,2)
Go straight for "sqrt(2) * 1 units"

>> Next one is (3,3)
Face 0 radians "Turn 45 degrees to Left"

>> Next one is (3,4)
Go straight for a unit
```

- ❏ So, Given 2 cartesian coordinates, we can figure out distance to move. This is the old dist function I implemented.

- ❑ We need to keep track of our global orientation. Then we can know if we need to turn or not.
    - ❑ Initially, face 0 radians.

Here is the code that works, but still needs testing

```
def path(size_of_cell, next_move):
      #next move is a cost_list
      path = []
      prev_angle = 0 #0 radians
      print next_move

      for i in range(len(next_move)-1):
            cur = next_move[i]
            nex = next_move[i+1]
            cur_angle =  angle(cur, nex)
            print cur, nex
            if prev_angle != cur_angle:
                  print cur_angle
                  print prev_angle
                  turn_angle = prev_angle - cur_angle
                  print turn_angle

                  orientation = "R"

                  if turn_angle < 0:
                        orientation  = "L"

                  if math.fabs(turn_angle) > 180:
                        turn_angle = 360 - math.fabs(turn_angle)

                  turn_angle = math.fabs(turn_angle)
                  prev_angle = cur_angle # this is what we
                  path.append((turn_angle, orientation))
                  d = dist(cur, nex) #This is how much we will move by
                  path.append(d * size_of_cell)
            else:
                  d = dist(cur, nex) #This is how much we will move by
                  path[-1] = path[-1] + d


      return path
```

TUESDAY 4/10

TO DO FOR TODAY

1. Implement the second cost function
2. Return path of points as “go straight, turn left by x degrees” commands

CONSIDERATIONS FOR LATER

3. This paper has some pseudocode that I think can help us https://pdfs.semanticscholar.org/e454/1b636c7a48d6bcbf030b455da8b37c8acaa7.pdf
4. Size of each pixel
5. Rotating (talked with Meba about this)

Implementing the second cost function

So I tried working on the function defined given on the Saranya paper for the 3rd time.
I am convinced that either the matricies are wrong, or that I am missing something.
Now, my plan is to come up with a reasonable function myself.
Cost = cost + abs(slope) * factor
This will need to be updated as we will choose the minimum one and leave pointers

Great Idea

We will use Hidden Markov Models on Dynamic programming tables to implement this matrix fast!
I am excited
I learnt about this in my Computational Biology class
YAY!

UPDATED COST FUNCTION PSEUDOCODE

1. DP = Start with an int_max matrix #cost table
2. BT = Start with an -1 matrix #backtracking table
    - Set the target cell a special character, like ‘*’
3. Use the following table for navigating to neighboring cells

3	2	1
4	I’m here	0
5	6	7

4. Filling up the next cell:
    - Do for all 8 neighboring cells: for x in range(8):
      If they exist (i.e. we are not at the corner
      #calculate x

Cost_of_moving_from_x_to_here =
10 (if x is even) #horizontal/vertical
20 (if x is odd) #diagonal
Temp = val_at_x (this is from the first cost matrix)
+ (slope_to_x * factor)
+ Cost_of_moving_from_x_to_here
IF temp < dp(current cell):
dp(current_cell) = temp
bt(current_cell) = x

5. Now, the question is which loop we put this function in, i.e. where do we start from exploring?
Start at target cell
Then do the 4-quadrant model as in the naive cost function
Then re-update the whole thing

FRIDAY 4/6

I checked with my Algorithms professor, Christine, that the way we are implementing this matrix is the optimal, so YAY!

We will deal with all slope calculations in the initial slope function.
Slope function:
Input: 2 points (p1,p2) where p1 is the one we are at, p2 is the goal point
* In the fill_matrix functions we call the slope(p1,p2) with all points to fill
Usually we update it so that it looks like
p2_val = p1_val + X
No we will have
P2_val = p1_val + X + slope(p1,p2)

Factors in slope function
We will return the absolute value function
But before that,
if the value calculated is negative, then we are descending from start to end → MULTIPLY BY 10
If val is positive, we are ascending from start to end → MULTIPLY BY 20
Take abs value
Return
We will add high-slope penalties here too, with big factors

IMPORTANT CONSIDERATIONS FOR THE CASE OF TILTING/MISORIENTATION
The size of each pixel: a brute way of doing it is to bind them so that each pixel is the size of the rover
Turning: The rover can't just turn x degrees, without moving some distance, so the path-to-directions function cannot just depend on the points…
Checking if we are at the correct point: After every point we "descend" or "turn more than x degrees" we expect a high chance of disorientation. Make the cost function depend on that or make another cost function→ then make checks at these points (small lidar scans)
Updating the cost function: Should we make it dependent on the degree to turn?

Again having trouble with slope function
Here is a worksheet

```
ELEVATION MATRIX
      0      1      2      3
0  0.0001 -0.0299 -0.1100  0.0000
1 -0.0088  1.8559 -0.2729  0.0130
2 -0.0137  0.2289  2.4338  0.0125
3  0.0000  0.1099  0.1107  0.0000
```

.0001	**-0.0299**		
-0.0088	**Cost 20** **descending** **Slope 42.576**	**Cost 10** **ascending** **Slope 5.718**	**Cost 0**

I couldn't get the same row to work yet, so I am moving on to the same column calculations

PROBLEM: I just can't…
So the cost2 matrix relates to the cost_2 matrix
But the prev calculations won't work because we can't just do horizontal/vertical
We need to traverse the matrix a second time, to get the minimum value for each…
Omg

IDEA: maybe I can do the quadrant calculation again???
So sad
Very sad

BETTER IDEA
Maybe just write a separate function that doesn't depend on the naive-cost function

Some code that once worked, but now omitted (pasting here in case I need it again)

```
#filing cols to left of goal
	for x in range(cost1.shape[1]): #shape[1] is num columns
		if g_col-x >= 1:
			# cost1[g_row, g_col-x-1] = cost1[g_row, g_col-x] + 10
			s = slope(Point(g_row, g_col-x, dem[g_row, g_col-x]), Point(g_row,
g_col-x-1, dem[g_row, g_col-x-1]))
			cost2[g_row, g_col-x-1] = cost2[g_row, g_col-x] + s + 10
			print cost2[g_row, g_col-x-1]
			#cost2[g_row, g_col-x-1] = cost1[g_row, g_col-x-1] +
slope(Point(g_row, g_col-x, cost1[g_row, g_col-x]), Point(g_row, g_col-x-1, cost1[g_row,
g_col-x-1]))
		if g_col-x<1: #until we hit 0 with g_col-x-1 = 0 i.e. until g_col-x = 1
			break
		'''print "\nCOST MATRIX"
		print DataFrame(cost1)'''

	#filling cols to right of goal
	for x in range(cost1.shape[1]): #shape[1] is num columns
		#until we hit 0 with g_col-x+1 = shape[1]-1 i.e. g_col-x = s[1]-2
		if g_col+x+1 <= cost1.shape[1]-1:
			s = slope(Point(g_row, g_col+x, dem[g_row, g_col+x]), Point(g_row,
g_col+x+1, dem[g_row, g_col+x+1]))
			cost2[g_row, g_col+x+1] = cost2[g_row, g_col+x] + s+ 10

		if g_col+x+1 > cost1.shape[1]-1:
			break

		'''print "\nCOST MATRIX"
		print DataFrame(cost1)'''
```

TUESDAY 4/3

FIXED: I am struggling with implementation of the 2D array, lots of bugs will happen when we run.
But I have to implement a buggy version for now to proceed on the rest of the algorithm.
Will come back to the cost function later on.

First thing I will work on is the recursive editing in fill_matrix function.

THIS WORKED:
Current code took the goal point to be on the bottom right corner of the matrix
I was struggling to do it otherwise for now
Resolved: Consider it to be quadrants around the goal point. We know how to build the 2nd quadrant. We can do the rest of the quadrants

TO DO

Before implementing the real cost function, I will try to output the path.. So that we have an idea of the output…

RESOLVED I am working on this right now, but we cannot append and update right away, before checking all possible neighboring cells :'(

I am thrilled to announce that our code can now figure out a path for the basic/naive cost function.

To Do After Implementing the Code

1. **Give like 1000 penalty for obstacles**
2. **To get a planar surface, subtract a plane and fit into topography**
3. **Do we have a critical slope? Put x1000 factor**
4. **We have a finite size, how can we fit?**

SUNDAY 4/1

I am in the process of implementing the algorithm described in this paper: https://www.sciencedirect.com/science/article/pii/S2405896316300490 .

Question: Should I turn the matrix into a point matrix? How can I do that?
Real Question: The output will be the points to follow. Is that OKAY?

HOW DOES THIS CODE WORK - Demo by examples

Cost of moving horizontal/vertical is 10, diagonal is 14
Ascending slope factor is 20, decreasing slope factor is 10
Take the minimum at every point to decide the next step

Matrix 0

The one used in the paper

Elevation:

St -0.11	**0**
-0.2729	**End 0.013**

Cost without slope

14	**10**
10	**0**

Cost with slope: cost = cost + abs(slope) * fact

14 + (0.11 + 0.013)/sqrt(2) * 20 = 15.7394	**10 + .013*20 = 10.26**
10 + (0.2729+0.013) * 20 = 15.718	**0**

Matrix 1

Start at 00, go to 11. Path: 00 → 11

1	**10**
3	**5**

The cost matrix without slope:
Path 00-11 costs 14
Path 00 - 10/01 - 11 costs 10+10=20

14	**10**
10	**0**

Cost matrix with slope

14	**10**
10	**0**

Matrix 2

Start at 00, go to 11. Path: 00 → 11

1	**10**

3	**11**

Matrix 3

Start at 00, go to 11. Path: 00 - 01 - 11

1	**10**
3	**11**

MONDAY 3/26 and TUESDAY 3/27

NEED TO LOOK UP: Topographic numeric array with elevation information

IMPORTANT NOTE: I decided to change to using D* algorithm, described as an algorithm that solves assumption based path planning problems, where a robot has to navigate to given coordinates in unknown terrain. This is more efficient than repeated A* searches. https://en.wikipedia.org/wiki/D* This was also used in Opportunity and Spirit!

Another link that verifies that D* is optimal: http://people.csail.mit.edu/rak/www/?q=node/14

AMAZING: A D* algorithm that incorporates terrain slopes to plan a path: https://www.sciencedirect.com/science/article/pii/S2405896316300490 Also quiet recently published!!!!

Here is the summary of my reading of this paper (Saranya et al., 2016)

D* (HOW IT WORKS)

The area of exploration is split into nxn grid. Each cell in the grid has a state to denote the presence of absence of obstacles.

Cost function: Distance to be travelled. Cost of moving along and across the call is the same. Moving to the diagonal (d)is additional cost.

Let p be the cost of moving horizontally (h) or vertically (v)

Then **cost(x,y)_h|v = p * units** and **cost(x,y)_d = p * sqrt(2) * units**

Slope function: Let A = (xa, ya, za) and B = (xb, yb, zb) be points. Then the slope between A and B is

slope(A,B) = (zb - za) / sqrt((xb-xa)^2 + (yb-ya)^2)

azimuth(A,B) = 180/ (pi * tan((xb-xa) / (yb-ya))

Modified cost function:

modified_cost(A,B) = cost(A,B) + abs(slope(A,B)) * Factor

Where factor is

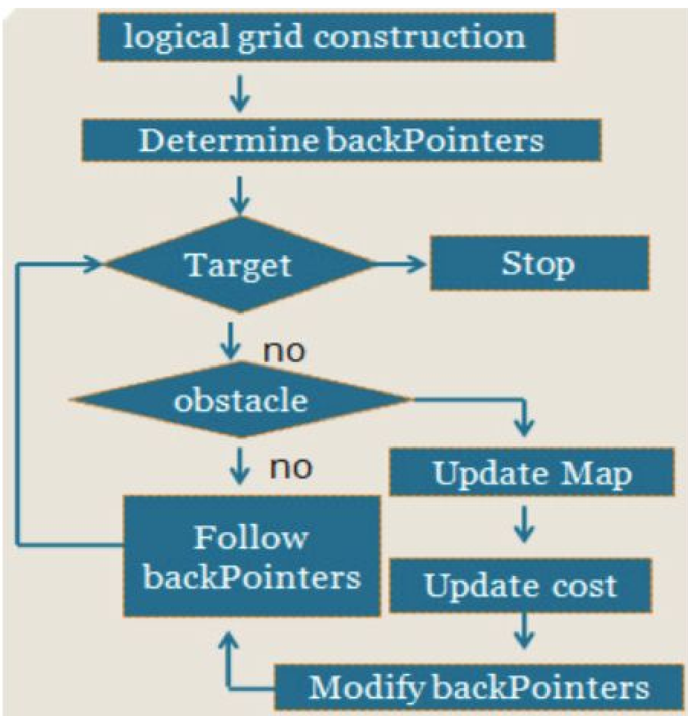
20 if ascending

10 if descending

0 if flat

I couldn't find an implemented version of the modified D* algorithm, but I found the pseudocode! It's attached below. So I will code it first.
Backtracking matrix etc will take time !

```
If hnew (Y,G) > k(Y)
   Obstacle detected
      If Y is in shortest path
         AssesReComp
      End if
End if

AssesReComp
  Replan=1
  For each neighbour X of Y:
    If  Cost(X,G) < k(Y)
       While X !=TargetCell
          If Bk(X) = walkable
             X=Bk(X)
             Replan = 0
          End if
       End While
       Next_cell=X
    End if
  End For
  If Replan = 1
       CostUpdate
  End If
End AssesReComp


CostUpdate
   recompute  h(Y,G)
   find k(Y)
   update bk(Y)
End CostUpdate
```



TUESDAY 3/20

Summary of Technical Challenges:
The LIDAR output data is in terms of spherical coordinates. We will have to turn them into angles and distances (Multivariable calc to rescue).
We are interested in not finding the best path only, but a few good paths so that, when we feed in a few interesting targets, we can find the best path among them.

Here is an example to demonstrate what I mean:

1. Ground station sends 3 targets T1, T2, T3, ranked from most interesting to least, with a point of importance. Let's say T1 (10), T2 (9), T3 (2)
2. Rover finds 2 paths to get to each target, P_{11}, P_{12}, P_{21}, P_{22}, P_{31}, P_{32}, so 6 paths in total. There is a score associated with easiness of each path, determined by steepness etc. (steeper and longer path = lower point, flatter and shorter path = higher point) so let's say P_{11} (18), P_{12} (30), P_{21} (12), P_{22} (25), P_{31} (40), P_{32} (8)
3. Now we multiple Tn points with Pn points, for all paths we found
4. We get P_{11} (180), P_{12} (300), P_{21} (98), P_{22} (225), P_{31} (80), P_{32} (16)
5. The path P_{12} (300) and P_{22} (225) are the most viable ones, so we will first visit T1 through the second path and later, time permitting, T2 through the second path.

What to Edit-v2:

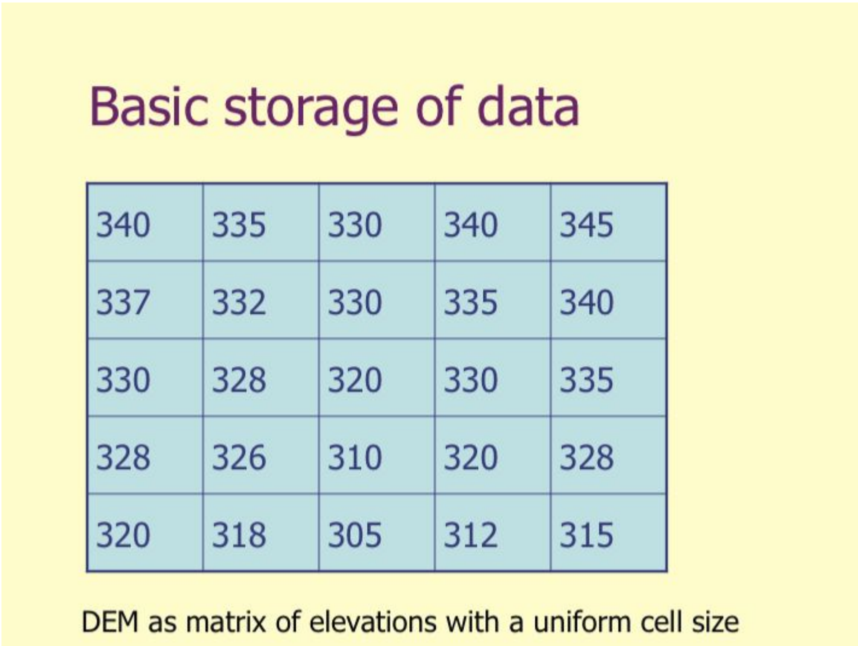
Look at Biased Random Walk algorithms
Wes recommended that we give penalty for certain paths etc.
https://en.wikipedia.org/wiki/Biased_random_walk_on_a_graph

This current code considers the best path on a flat surface. I will add the elevation to this code.
Given a point, we have the x,y,z information for every point. We will calculate the trig for elevation from **raster dataset**
I am not sure if this is what we want, but this is the definition: "A **raster** consists of a matrix of cells (or pixels) organized into rows and columns (or a grid) where each cell contains a value representing information, such as temperature. **Rasters** are digital aerial photographs, imagery from satellites, digital pictures, or even scanned maps."
Here is information on Digital Elevation models:
https://gisgeography.com/dem-dsm-dtm-differences/

This to-do list is cancelled because I am now using the D* instead of A*
I will make the demo show more paths (ie. not the best one only, maybe 2 for now)
Create fake input
- This is called a **raster dataset**

Comment on the astar.py so it's better understandable
Comment on astar_demo.py
Make the astar.py return all the possible paths as a list
Make the display astar_demo.py display first 2 in different colors
Iterate through the pathlists and display them
Make the output of demo file a list of points, so it is easier to communicate to motors
Turn input data into a topographic map
- Here all # barriers will be converted to a number, higher than 10, let's say.

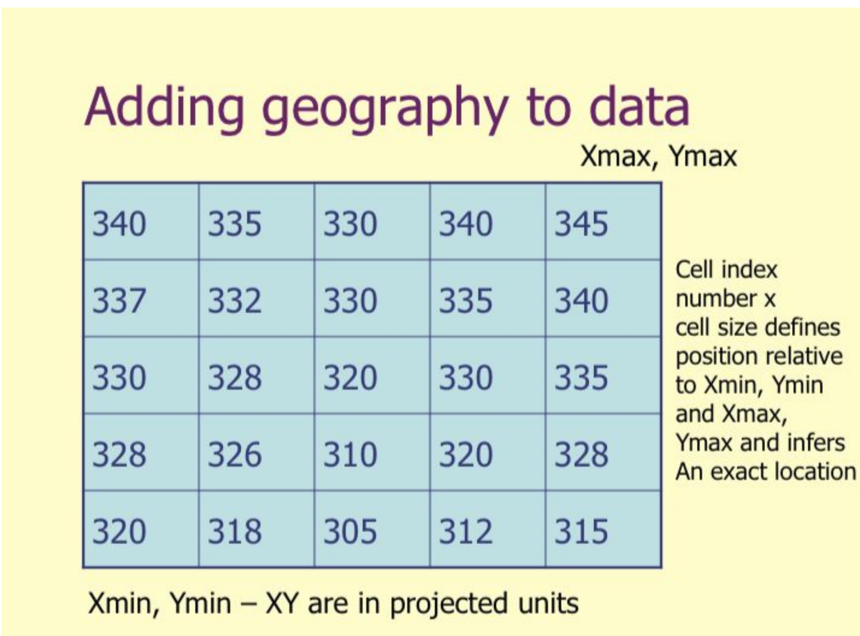
More on creating the fake input (From https://libraries.mit.edu/files/gis/DEM.pdf)



Basic storage of data

340	335	330	340	345
337	332	330	335	340
330	328	320	330	335
328	326	310	320	328
320	318	305	312	315

DEM as matrix of elevations with a uniform cell size



Adding geography to data

Xmax, Ymax

340	335	330	340	345
337	332	330	335	340
330	328	320	330	335
328	326	310	320	328
320	318	305	312	315

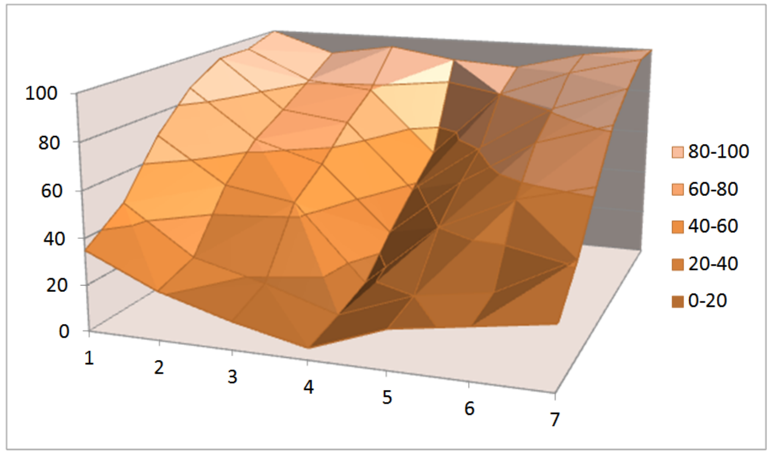
Cell index number x cell size defines position relative to Xmin, Ymin and Xmax, Ymax and infers An exact location

Xmin, Ymin – XY are in projected units

Mapping from DEM: (Source:**https://serc.carleton.edu/details/images/36309.html)**

100	90	95	90	88	96	100
95	81	78	49	80	92	100
95	72	68	38	61	81	92
86	64	55	26	52	72	82
70	50	45	12	40	55	63
47	26	18	8	20	25	42
35	21	12	5	17	22	27

(a)



(b)

MONDAY 3/19

Sketch of flow chart
Rover start ⇒ Take images + LIDAR ⇒ Package and send Stereo camera image + LIDAR =Radio=> unpackage + decorolate LIDAR + Image ⇒ (1) Lidar used for topo Map (2) team picks n areas of interest and rank them, send those (x,y,z) points relative to rovers originals position =Radio=> rover figures out the path =acquire imagery, spectra/stereo images/lidar of whole outcrop

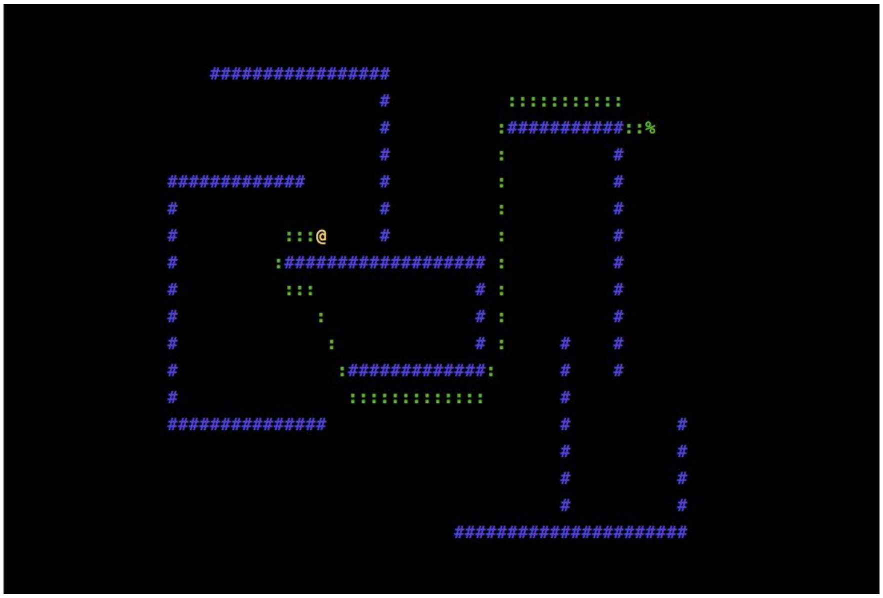
Sketch of software flowchart

1. Align images and lidar together
2. Format of candidate
3. Algorithm

Looking into this PathFinding Python library
https://github.com/brean/python-pathfinding/blob/master/README.md
http://qiao.github.io/PathFinding.js/visual/

This is the A* code I will be editing
https://github.com/elemel/python-astar/tree/master/src

This comes with a simulator, here is a terminal screenshot of simulator:



What to Edit-v1:

<u>This current code considers the best path on a flat surface. I will add the elevation to this code.</u>
Given a point, we have the x,y,z information for every point. We will calculate the trig

<u>I will make the demo show more paths (ie. not the best one only, maybe 2 for now)</u>
DONE - Make the astar.py return all the possible paths as a list
DONE - Make the display astar_demo.py display first 2 in different colors
DONE - Make astar_demo accept lists as an input
DONE - Iterate through the pathlists and display them

SUNDAY 3/18

<u>Wes' feedback on the proposal:</u>
for drive planning, how will the mast camera-captured image space be mapped onto the LIDAR topography? i.e., you want to identify cool places to explore in an image: how do you find that same location in the lidar-derived topography? (need a map of image pixels to -> alt/az : a cool challenge; can be done with a calibration target)

FRIDAY 3/16

Reading some papers to get a better sense of the work that's been done:

- http://ieeexplore.ieee.org/abstract/document/5548134/
    - This paper talks about LIDAR based road-edge detection
    - Here is the abstract:
        - In this paper, a LIDAR-based road and road-edge detection method is proposed to identify road regions and road-edges, which is an essential component of autonomous vehicles. LIDAR range data is decomposed into signals in elevation and signals projected on the ground plane. First, the elevation-based signals are processed by filtering techniques to identify the road candidate region, and by pattern recognition techniques to determine whether the candidate region is a road segment. Then, the line representation of the projected signals on the ground plane is identified and compared to a simple road model in the top-down view to determine whether the candidate region is a road segment with its road-edges. The proposed method provides fast processing speed and reliable detection performance of road and road-edge detection. The proposed framework has been verified through the DARPA Urban Challenge to show its robustness and efficiency on the winning entry Boss vehicle.

TUESDAY 3/13

There are a few algorithms that robotics use widely:

- If we have a simple map and simple motion laws for the rover, typically A* algorithm works for shortest path finding.
    - Pathfinding algorithms involve planning in advance
    - Motion + detection algorithms involve incorporating the sensor information when we hit something
    - http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

- Reducing the map to a graph
    - We want to reduce the LIDAR point-cloud to a **weighted, directed graph**.
    - How to represent the obstacles in the graph? 3 options as described: https://www.redblobgames.com/pathfinding/grids/graphs.html
        - Infinite weight
        - Remove edges
        - Remove nodes
- http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html#the-a-star-algorithm
- https://muhaimenshamsi.wordpress.com/2015/08/07/navigation-using-lidar/

MONDAY 3/12

Research about the data LIDARs produce -- Point Cloud
For example, point cloud is obviously a vector based structure - each point has its XYZ coordinates, and some attributes called **components** in FME. Components can represent time, flight line, intensity (how much light returns back from a point), color, etc.

There are some free point cloud data out there, but they are in .las format, and my Mac refuses to open them. This thread contains them:
https://gis.stackexchange.com/questions/18202/seeking-point-cloud-lidar-data

Do we have access to ArcGIS, the mapping software? This website seems quiet useful:http://desktop.arcgis.com/en/arcmap/10.3/manage-data/las-dataset/what-is-a-las-dataset-.htm

https://www.liblas.org/start.html#

Friday 3/9

Some research on Path-finding algorithms:

First you'd need an exploration algorithm to find the goal. Second you'd need an optimization algorithm to figure out the shortest known path home. In the video, they use the left hand wall exploring algorithm to explore the unknown map, and either an A* or Dijkstra search to find their way back to the start.

Maze solving algorithm
A* search algorithm

To put this into graph notation, nodes are squares, walls are edges.

To implement the algorithms you'll need to be able to represent each of three possible states for each edge (unexplored, wall, free) as well as a minimum cost and next node (for A*/Dijkstra). You could represent that with just 224 bits in a 4x4 maze. Memory shouldn't be an issue.

Tuesday 3/6

If we do a ML approach, we will train the model on MacBook, and then **pickle** the file to transfer to Raspberry Pi. (using scp command)
Then we classify.

PLAN after visiting the outcrops

1- Come up with a path-finding algorithm

2- How do we detect the slopes
Accelerometer knows its tilted (10 = normal, measure downwards etc.)

3- LIDAR images - get the z axis

Algorithm idea:

Avoid the higher z axes
Incorporate the size of the rover (think of it as a cube)
IR sensors
Come up with the best path to the target

We want to make it semi-autonomous,
which means, we will give the rover some decision making abilities

The robots will control mobility by sensor inputs.

Our inputs are the LIDAR and IR sensors
With LIDAR we will produce a 3D map
From what the LIDAR sees, we have x,y,z points

Once we determine the target outcrop we want rover to investigate, we will communicate the target to rover
The rover will decide which path to use to reach the target with an algorithm
The algorithm depends on the steepness of the slopes, the narrowness of paths

The path-finding problem is a well studied algorithmic problem. There are greedy algorithms, which operate on finding the optimal path at every step and non-greedy algorithms which look at the general picture and find a path for the whole time.

Friday 3/2

An interesting idea: iPhone controlled robot
https://maker.pro/projects/raspberry-pi/raspberry-pi-robot

The lidar has obstacle avoidance capabilities
We will add the IR sensors
so that we have a vision of objects that are in LIDAR's blindside

What the machine sees: for every point in space, we will see x,y,z points
we are at point (0,0,0)

- avoid paths narrow for us
- avoid branches on top of us
- look at IR + LIDAR in conjuction
- slopes (that are short/tall for us)