Things of the Internet (TOI)

LAB 1: RF Based Positioning

SAMPLE SOLUTIONS

Please note the submission deadline for this report is Tuesday 12:00 PM, March 23rd, 2021.

Your name:

Introduction

Localization is a key capability for things of the internet. Things need to know where they are to provide context to sensor readings. There are many ways of working out where a sensor is, ranging from GPS to inertial measurement units. However, one of the simplest techniques is to use measurements of ambient radio signal strength e.g. WiFi. Things are equipped with wireless transceivers anyway for communication, so the key is to try and exploit the transceiver to give measurements of location. This is particularly well suited to indoor settings, where a number of APs (access points/routers) are installed to provide good communication coverage so mobile devices can connect to a wireless network. The operating principle is simple: the closer the device is to an AP, the stronger the received signal strength is. However, the devil is in the details – the surrounding environment (walls, people, furniture, obstacles etc etc) all conspire to alter the relationship between signal strength and distance, so what starts off as a nice straight line in theory, turns into a very wiggly and noisy relationship. What is worse, this relationship is dynamic as well – a stationary WiFi device will typically experience a variation in signal strength over time.

So the question is: how are we going to use these noisy measurements to estimate a device's location and how accurate can it be? Rather than using a physics based model and trying to model and represent all the sources of disturbance, we take a much simpler approach. We approach the problem by breaking it down into two stages. In the first stage (the offline phase), we manually survey the signal strengths, i.e. we stand in a surveyed location and record what APs can be heard and with what strength. We obtain a vector (tuple) of measurements e.g. <X:-20,Y:-30,Z-32> could be the vector from one location, where we can hear APs X,Y and Z and <W:-40,Z-60> could be the vector from another location where we can hear APs W and Z. These vectors of signal strength are commonly called fingerprints i.e. the hope is that they are sufficiently unique and discriminative such that every location has a different fingerprint. Obviously, there is a time (and consequently financial) cost in building the map, so there is a question about how precisely the map needs to be surveyed e.g. on a 1 m grid spacing or on a 5 m grid spacing or even simply in the centre of each room.

In the second phase (the online phase), the device will record a vector of signal strengths. The goal now is to come up with some algorithm that, with the aid of the map collected in the offline phase, could accurately determine the location of the device. There are many different approaches to doing this, and the aim of this lab is to demonstrate how a location system could be built using WiFi signal strength measurements, to consider factors which would impact its performance, and to discuss its relative merits.

> **NOTE** The labs in this course are based on Python 3.7. Please refer to the environment setting introduction for details. If you have any problems with Python version please ask for help.

Exploratory Data Analysis

We have collected a set of data for use in this lab. During the offline phase we survey the environment collecting RSS measurements. For this demonstration we have collected 60 samples per location. Navigate to “data/set1/wifiData/” and open “Wifi_14.txt” with a text editor to look at a sample file. The first few lines look like the following:

```
 2 4
1 AP_1 -37.89
2 AP_2 -52.63
3 AP_3 -44.98
4 AP_1 -38.68
5 AP_2 -61.14
6 AP_3 -35.08
7 AP_1 -39.46
8 AP_2 -61.62
9 AP_3 -47.00
10 AP_1 -38.00
11 AP_2 -60.69
.....
```

The first line in the file is the survey coordinate i.e. (x = 2 m, y = 4 m). The remainder of the lines detail the signal strengths in dBm from each access point. Note that each access point normally has a long BSSID as a unique identifier (e.g. `d8:c7:c8:cc:43:24`), but we have made things simpler here for ease of understanding.

Each file corresponds to a different survey location and typically 60 measurements are taken from each AP. In this dataset (set1), there are 3 APs in range, and in total we have collected Wi-Fi samples from 121 locations.

During the online phase we want to estimate the user’s unknown location using RSS readings collected at that location. Navigate to “data/set1/testWifiData/” and answer the following questions:

Q: What is the number of unknown locations that we are going to estimate?

Your answer:

63

Q: How many RSS measurements have we collected per location per access point?

Your answer:

50

Q: What do you see in the first line of each file? Comment on that.

Your answer:

Ground Truth

Finally, we have collected two datasets (i.e. set1 and set2); set1 contains RSS measurements with respect to 3 APs and set2 with respect to 5 APs.

Loading and visualizing

We have written some helper functions which load the datasets (both test and train) and visualize their spatial distribution, which can be found in `helper.py`.

In [1]:
```
import helper as myhelper
```

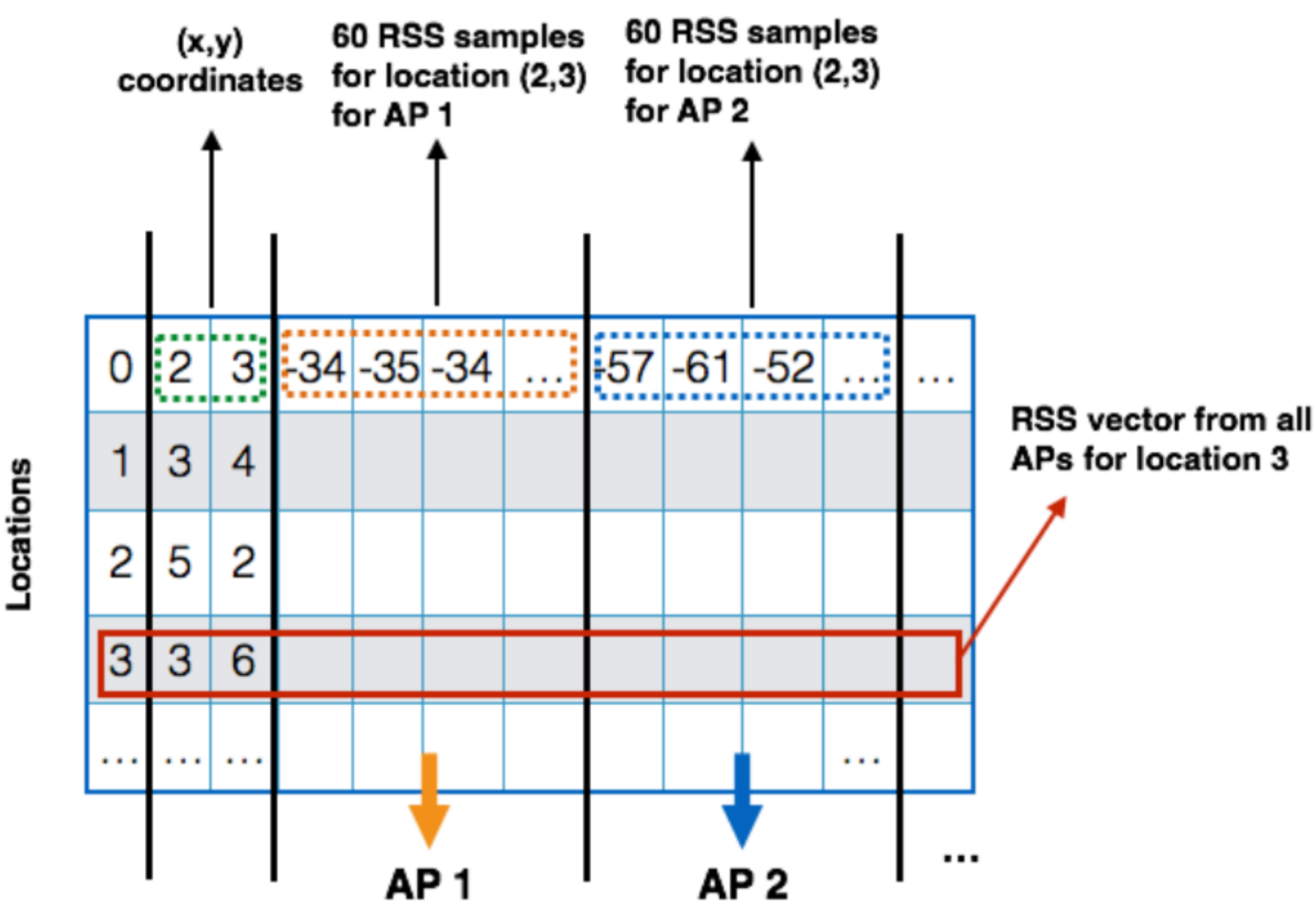
The function `load_wifi_data` has the following API

```
def load_wifi_data(data_folder, n_samples, n_ap):
    """
    :param data_folder: Location of WiFi RSS data
    :param n_samples: Number of RSS samples to return for each location
    :param n_ap: Number of access points
    :return: train and test wifi databases
    """
```

Lets load up the train and test datasets from Set1 (3 APs).

In [2]:
```
import os
parent_dir = os.path.abspath(os.path.join('..'))
datasetpath = os.path.join(parent_dir,'lab1','data','set1')
NUMBER_OF_APS = 3
MAX_SAMPLES = 60
trainData,testData = myhelper.load_wifi_data(datasetpath,MAX_SAMPLES,NUMBER_OF_APS)
```

The dataset is stored simply as a large array, with the following structure.



Q: Do you think this is the best representation of the dataset? What would be a better (and indeed more Pythonic) way?

Your answer:

> Using a dictionary would be far more elegant.

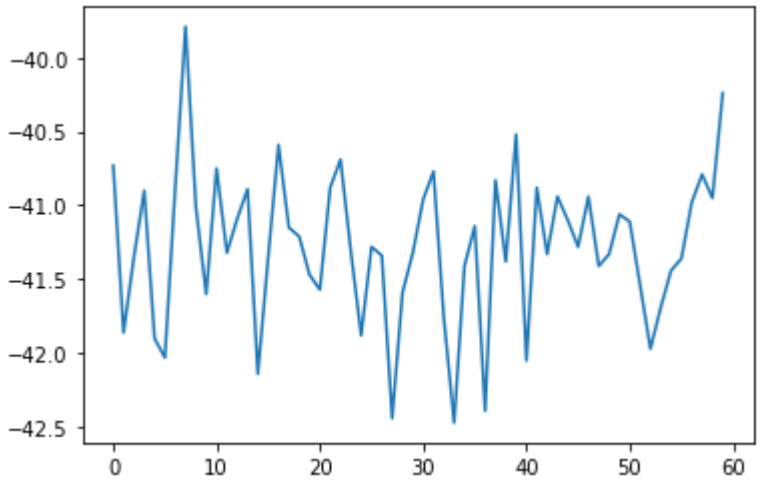
Lets look at the first row of the training dataset. It corresponds to the readings obtained from the location (0,0)

In [3]:
```
# look a the first row of the array
trainData[0,:]
```

Out[3]:
```
array([  0.  ,   0.  ,   0.  , -44.63, -44.09, -43.74, -43.69, -44.37,
       -44.3 , -44.96, -44.57, -46.01, -43.99, -44.6 , -44.89, -43.89,
       -44.53, -45.14, -44.4 , -44.32, -44.19, -44.41, -44.92, -43.79,
       -45.79, -44.27, -44.32, -44.67, -44.83, -44.23, -44.29, -44.55,
       -44.9 , -44.84, -45.3 , -44.42, -44.97, -44.95, -44.8 , -44.34,
       -44.69, -43.8 , -44.66, -45.76, -44.31, -44.98, -44.3 , -43.9 ,
       -44.34, -43.64, -44.51, -44.48, -43.19, -44.48, -44.25, -45.33,
       -44.09, -44.75, -44.09, -44.73, -44.47, -43.95, -43.95, -70.63,
       -56.39, -59.67, -58.89, -64.99, -55.39, -62.72, -62.12, -59.01,
       -66.36, -58.59, -71.66, -59.06, -63.67, -63.53, -63.44, -54.89,
       -62.46, -66.25, -53.15, -49.2 , -64.24, -61.71, -57.89, -60.21,
       -62.89, -65.15, -59.22, -56.98, -63.78, -57.22, -60.62, -65.19,
       -64.11, -65.38, -68.56, -68.29, -62.27, -67.61, -61.4 , -59.8 ,
       -50.59, -66.38, -73.23, -60.35, -64.27, -61.51, -70.02, -60.91,
       -62.85, -57.52, -68.24, -59.05, -61.  , -60.25, -66.18, -69.91,
       -60.6 , -56.54, -60.83, -41.42, -51.33, -35.13, -53.4 , -39.19,
       -37.62, -46.27, -51.46, -43.91, -40.82, -49.55, -42.25, -43.89,
       -40.04, -41.11, -39.11, -43.09, -45.77, -39.88, -40.78, -43.56,
       -40.52, -42.42, -51.24, -49.31, -46.33, -40.37, -42.45, -43.32,
       -47.38, -33.73, -39.82, -44.02, -44.78, -39.36, -39.71, -47.53,
       -43.93, -51.54, -41.04, -43.84, -47.58, -47.22, -41.41, -35.47,
       -45.26, -39.6 , -38.42, -45.08, -40.46, -41.88, -46.54, -41.63,
       -43.96, -48.2 , -44.18, -33.65, -44.12, -46.  , -48.89])
```

Investigate the sample strengths from AP1 at this location point, using the snippet below.

In [7]:
```
import pylab
import numpy as np
signalStrengthAP1 = trainData[3,3:MAX_SAMPLES+3] # We use 3 as an offset to handle the location/file-info in the array.
pylab.plot(signalStrengthAP1)
pylab.show()
```



Q: What do you notice about the signal strength readings?

Your answer:

Not stable, fluctuates in an interval. This is because of random variations e.g. multipath or noise in the radio channel itself.

Q: Change the location index and examine what happens to the signal strength. Is there any pattern you can see?

Your answer:

> The value fluctuates around a certain value

Q: Can you estimate the likely position of AP1?

Your answer:

> (6,6)

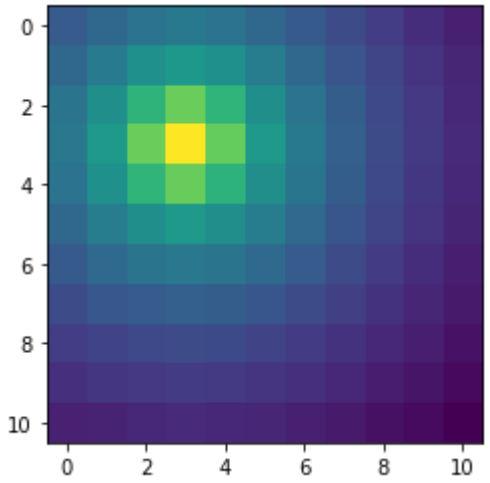
Advanced question: (optional) Can you visualize 2-D map of signal strength for each access point?

Your answer:

In [8]:
```
import numpy as np
heatmap = np.zeros((11,11))
for i in range(0,121):
    heatmap[int(trainData[i,1]/2)][int(trainData[i,2]/2)] = np.mean(trainData[i,3:MAX_SAMPLES+3])
print(heatmap)
pylab.imshow(heatmap)
```

```
[[-44.4925      -43.02433333 -41.87616667 -41.27216667 -41.91066667
  -43.05866667 -44.54133333 -45.93983333 -47.38483333 -48.72266667
  -49.86766667]
 [-43.02716667 -40.88416667 -38.53133333 -37.71616667 -38.62083333
  -40.70483333 -43.1425      -45.06466667 -46.60166667 -48.16083333
  -49.39433333]
 [-41.7395      -38.66516667 -34.49316667 -31.246       -34.45233333
  -38.721       -41.84366667 -44.20933333 -46.269       -47.8075
  -49.20283333]
 [-41.3505      -37.5695      -31.235       -24.95        -31.37833333
  -37.62316667 -41.24266667 -43.92916667 -45.9815      -47.67633333
  -49.064      ]
 [-41.8495      -38.554       -34.455       -31.33216667 -34.58566667
  -38.81        -41.694       -44.262       -46.263       -47.847
  -49.17066667]
 [-43.07683333 -40.806       -38.703       -37.664       -38.83083333
  -40.79966667 -42.9715      -44.99        -46.6035      -48.18366667
  -49.4375     ]
 [-44.54783333 -42.95466667 -41.82433333 -41.40583333 -41.76066667
  -43.0075      -44.40966667 -46.01        -47.47466667 -48.75666667
  -49.99266667]
 [-46.04816667 -44.9045      -44.309       -43.95183333 -44.22316667
  -44.99183333 -46.03966667 -47.07766667 -48.27916667 -49.33216667
  -50.35983333]
 [-47.3555      -46.791       -46.27283333 -45.97733333 -46.19183333
  -46.73983333 -47.438       -48.34983333 -49.21083333 -50.04266667
  -51.01516667]
 [-48.62916667 -47.99483333 -47.752       -47.57966667 -47.714
  -48.18633333 -48.663       -49.13783333 -50.18866667 -50.72083333
  -51.59516667]
 [-49.795       -49.55216667 -49.148       -48.99966667 -49.22233333
  -49.42333333 -49.91316667 -50.38183333 -51.10583333 -51.5385
  -52.31333333]]
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x227ff800588>
```



A simple approach to finding location

Now that we have a better understanding of how the signals vary over space (and time), we can now try and solve the localization problem: given a set of signal strengths at an unknown location and an offline map, can we find the best estimate of the device position?

One of the simplest approaches is just a nearest neighbour lookup i.e. we just compare our candidate set of signal strengths to the signal strengths in the offline map and find the closest match.

Q: What would be a good way of working out how "far" two sets of signal strengths are from each other?

Your answer:

e.g.: Euclidean Distance of mean value of both sets. Or perhaps simply `abs(mean(RSSI_1) - mean(RSSI_2))`

Rather than using all the raw information in the table, it perhaps would be more helpful to turn the 60 readings from each AP into representative statistics.

Q: What are some sensible statistics that could be used?

Your answer:

> There are many options here. A simple, yet robust option is to use the mean or median as a measure of centrality and the interquartile range (difference between top 75% and bottom 25% measurements) as a measure of spread.

> An alternative is to model (e.g. with a Gaussian/Normal distribution or similar) the centre and spread of the measurements.

In this approach, we will simply compute the mean of the readings in the map for each AP. We will then compute the difference between the trial signal strength and each of the values in the map, recording the index of the smallest distance. Lets start from the first point in the testData set.

Q: What is the ground-truth location of this point?

Your answer:

> The first line of the file testRow[1:3]

Fill in the skeleton code below to find the closest point in the offline map to the candidate point. Note that although we take 50 samples in the test data set, we are just using the first sample point i.e. an instantaneous snap-shot.

In [34]:
```
import numpy
def predict(i):
    # Take out the corresponding row out of the testData set
    testRow = testData[i,:]
    # Extract the ground-truth location
    groundTruthX = testRow[1]
    groundTruthY = testRow[2]
    # Take the Signal Strengths of each AP in the trial point
#     testAP1 = testRow[3]
#     testAP2 = testRow[3+50]
#     testAP3 = testRow[3+50*2]
    testAP1 = numpy.mean(testRow[3 : 3 + 50])
    testAP2 = numpy.mean(testRow[3+50 : 3+50*2])
    testAP3 = numpy.mean(testRow[3+50*2 : 3+50*3])

    lik = numpy.zeros(len(trainData))
    predicted_loc = np.zeros(2)
    # Iterate through the map
    for k in range(numpy.shape(trainData)[0]):
        surveyAP1mean = numpy.mean(trainData[k,3:3+MAX_SAMPLES])
        surveyAP2mean = numpy.mean(trainData[k,3+MAX_SAMPLES:3+2*MAX_SAMPLES])
        surveyAP3mean = numpy.mean(trainData[k,3+2*MAX_SAMPLES:3+3*MAX_SAMPLES])
        # compute the score here, and record the best position in the map
        dist = numpy.sqrt((testAP1-surveyAP1mean)**2+(testAP2-surveyAP2mean)**2+(testAP3-surveyAP3mean)**2)
        lik[k] = dist
        # Find the most likely location
    ind = np.argmin(lik)
    predicted_loc[0] = trainData[ind, 1]
    predicted_loc[1] = trainData[ind, 2]
    return predicted_loc

# print(predict(0))
# print(testData[0, 1:3])
```

Q: For the first testing data, modify the above code to find the closest match. What is the distance error between the ground-truth location and the closest match in the map?

Your answer:

The predicted location is [4, 10], the actual location is [3, 3].

Q: Now we try using the average value of the 50 samples in the data set instead of just the first one. Does the accuracy improve?

Your answer:

> Yes.

> Other options are: using median value, using mean value, averaging top k, or weighted average of top k, etc.

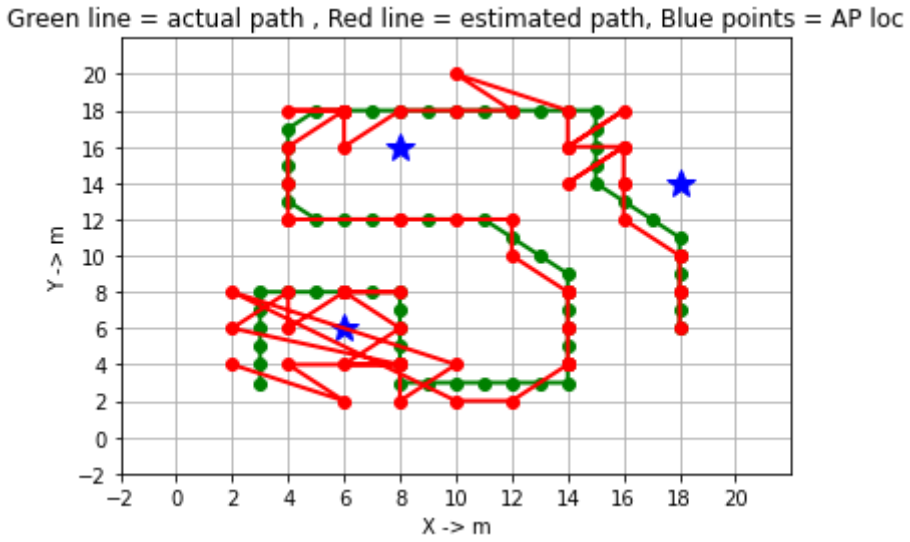
Q: Wrap your code into a function `predict()` that you can use to run through all the test points to compute errors for each test location. Visualize the result with the help of the following plot function. What is the mean overall error? What is the worst case error? Which do you think is more important for a location system and why?

Your answer:

In [35]:
```
# Plot the actual and estimated locations
import matplotlib.pyplot as plt
def plot_path(actual_loc, predicted_loc, n_ap):
    """
    :param actual_loc: Actual locations
    :param predicted_loc: Estimated locations
    :param n_ap: Number of APs
    :return: null
    """
    print("-> Visualizing estimated trajectory...")

    if n_ap == 3:
        ap_loc = np.array([[6, 6], [8, 16], [18, 14]])
    else:
        ap_loc = np.array([[6, 6], [8, 16], [18, 14], [16, 4], [12, 10]])

    plt.figure()
    plt.plot(ap_loc[:, 0], ap_loc[:, 1], 'b*', markersize=15)
    plt.plot(actual_loc[:, 0], actual_loc[:, 1], 'go-', linewidth=2)
    plt.plot(predicted_loc[:, 0], predicted_loc[:, 1], 'ro-', linewidth=2)

    major_ticks = np.arange(-2, 22, 2)

    axes = plt.gca()
    axes.set_xlim([-2, 22])
    axes.set_ylim([-2, 22])
    axes.set_xticks(major_ticks)
    axes.set_yticks(major_ticks)
    plt.xlabel('X -> m')
    plt.ylabel('Y -> m')
    plt.title('Green line = actual path , Red line = estimated path, Blue points = AP loc')
    plt.grid(True)
    plt.show(block=False)

    print("done\n")
    return

#your code here
actual_loc1 = testData[:, 1:3]
predicted_loc1 = np.zeros((len(testData), 2))
for i in range(len(testData)):
    predicted_loc1[i] = predict(i)
#predicted_loc = # your code here
plot_path(actual_loc1,predicted_loc1,3)
```

```
#calculate mean error, median error, etc. here.
```

```
-> Visualizing estimated trajectory...
```




```
done
```

In [37]:
```
maximum_err = 0
total_err = 0
for i in range(0,len(actual_loc1)):
    err = numpy.sqrt((actual_loc1[i,0]-predicted_loc1[i,0])**2+(actual_loc1[i,1]-predicted_loc1[i,1])**2)
    if err > maximum_err:
        maximum_err = err
    total_err += err
avg_err = total_err/len(actual_loc)
print("maximum error: ",maximum_err)
print("mean error: ",avg_err)
```

```
maximum error:  9.433981132056603
mean error:  1.4210947498468083
```

Q: (optional) Instead of finding the best (closest) match in the map, what if you instead found the top-k matches and then computed their mean? Do this centroid-weighting approach help or not?

Your answer:

depends on your algorithm, but typically it leads to more robust estimates

HORUS: A probabilistic approach to location finding

Instead of just taking the mean value of the signal strengths in the map, we can consider including higher-order statistics as well, e.g. the variance (or standard-deviation) that reflect the variability in the signal over time. One of the first approaches to wifi fingerprinting proposed just this technique, which we will implement a simplified version of.

> Youssef, Moustafa, and Ashok Agrawala. "The Horus WLAN location determination system." Proceedings of the 3rd international conference on Mobile systems, applications, and services. ACM, 2005.

Mathematical Model

Let X be a 2-dimensional physical space. At each location $x \in X$, we can measure the signal strength from k access points. We denote the k-dimensional signal strength space as S. We denote samples from the signal strength space S as s. We also assume that the samples from different access points are independent. The problem becomes, given a signal strength vector $s = (s_1, s_2 \dots s_k)$, we want to find the location $x \in X$ that maximizes the probability $\mathrm{P}(x|s)$

1) *Offline phase:* During the offline phase, the Horus system estimates the signal strength histogram for each access point at each location. These histograms represent the Horus system's radio map.

2) *Online phase:* Given a signal strength vector $s = (s_1, s_2 \dots s_k)$, we want to find the location $x \in X$ that maximizes the probability $P(x|s)$, i.e. we want to find

$$\mathrm{argmax}_x \, \mathrm{P}(x|s)$$

Using Bayes' theorem, this can be shown to be equivalent to:

$$\mathrm{argmax}_x \, \mathrm{P}(x|s) = \mathrm{argmax}_x \, \mathrm{P}(s|x) = \mathrm{argmax}_x \prod_{i=1}^{k} \mathrm{P}(s_i|x)$$

Effectively, it says that we should find the location in the map that would maximize the likelihood (probability) of resulting in the sampled measurements.

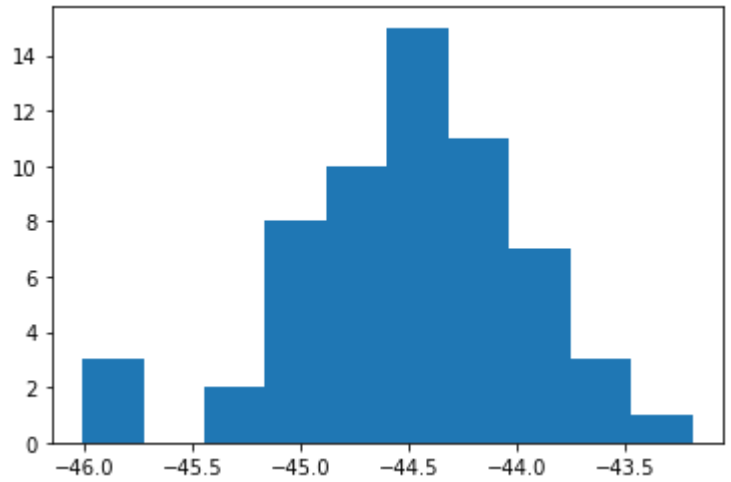
Offline Phase: Build a histogram map

In the offline phase, instead of storing N different raw measurements, or aggregating them using their mean, we will instead look at their distribution. Lets first investigate how best to represent the distribution compactly.

Q: Plot a histogram (hint: use `pylab.hist()`) of signal strengths from one AP for one location. What do you notice about the distribution? Does this hold for other points in the survey dataset?

In [26]:
```
#your code here
pylab.hist(trainData[0,3:MAX_SAMPLES+3])
```

Out[26]:
```
(array([ 3.,  0.,  2.,  8., 10., 15., 11.,  7.,  3.,  1.]),
 array([-46.01 , -45.728, -45.446, -45.164, -44.882, -44.6  , -44.318,
        -44.036, -43.754, -43.472, -43.19 ]),
 <BarContainer object of 10 artists>)
```



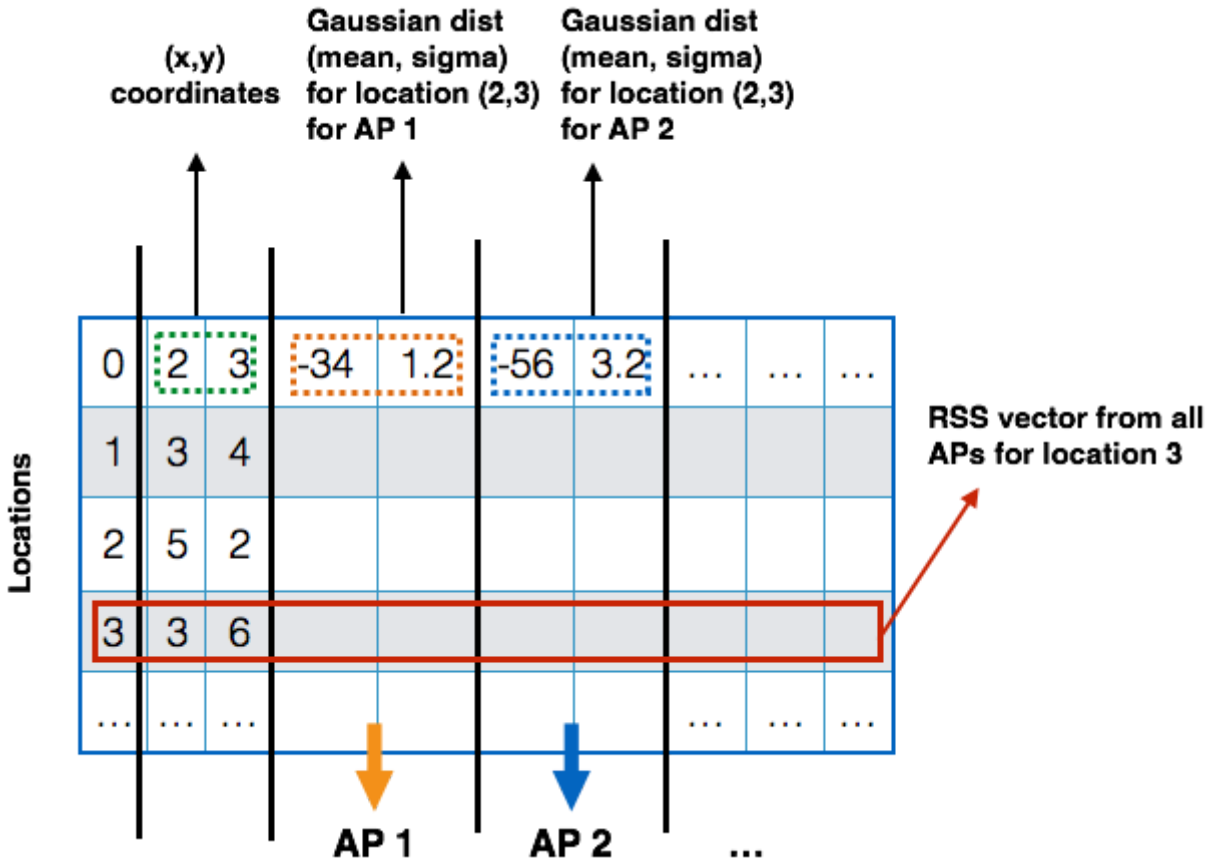
Your answer:

It looks like a normal (Gaussian) distribution.

Rather than storing the histogram as a set of bins, we can represent it as a normal (Gaussian) distribution. The normal distribution is parameterized by a mean μ and standard deviation σ. We can fit a normal distribution to a histogram of signal strengths to compactly represent the random variations, regardless of the number of samples in the histogram. We can use the helper function with the API below to transform the raw signal strength databases into normally distributed data.

```
def fit_data(train_db, n_samples, n_ap):
    """
    :param train_db: RSS points collected at known locations
    :param n_samples: Number of RSS samples per location
    :param n_ap: Number of access points
    :return: Wifi fingerprint database; We approximate the RSS at each location with a Gaussian
    """
```

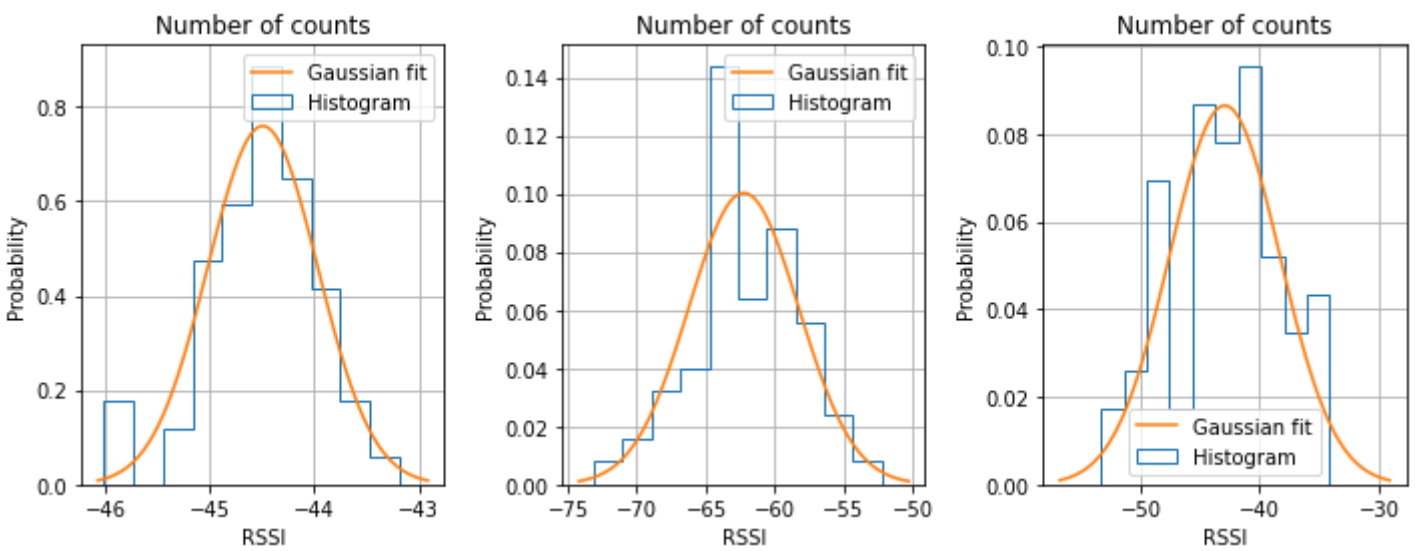
Given a raw dataset of signal strengths as above, this transforms it into an array with the following structure:



This is accomplished in the two lines of code below. We also visualize a few samples to have a better understanding of the norm fitting process.

In [27]:
```
normTrainData = myhelper.fit_data(trainData,MAX_SAMPLES,NUMBER_OF_APS)
normTestData = myhelper.fit_data(testData,MAX_SAMPLES,NUMBER_OF_APS)

# visualization for a better understanding
# visualize location 0 AP1, location 1 AP 2 and location 2 AP 3
import matplotlib.pyplot as plt
from scipy.stats import norm
%matplotlib inline
# Plot
fig, axs = plt.subplots(1,3, figsize=(10,4))

for measurement_idx in range(3):
        #Plot histogram
        n, bins, patches = axs[measurement_idx].hist(trainData[measurement_idx,
                                                    3+MAX_SAMPLES * measurement_idx: 3+MAX_SAMPLES*(measurement_idx + 1)],
                                                     bins=10, density=True, histtype='step', cumulative=False,
                                                     label='Histogram')

        #Compute and plot normal distribution
        mu = normTrainData[measurement_idx, 3+measurement_idx*2]
        sigma = normTrainData[measurement_idx, 4+measurement_idx*2]
        x = np.linspace(mu - 3*sigma, mu + 3*sigma, 100)
        axs[measurement_idx].plot(x, norm.pdf(x, mu, sigma),label='Gaussian fit')

        #Set labels
        axs[measurement_idx].grid(True)
        axs[measurement_idx].legend()
        axs[measurement_idx].set_title('Number of counts')
        axs[measurement_idx].set_xlabel('RSSI')
        axs[measurement_idx].set_ylabel('Probability')
plt.tight_layout()
plt.show()
```

```
-> Modeling RSS with Gaussian dist...done
-> Modeling RSS with Gaussian dist...done
```



Q: What are the pros and cons of using the Gaussian distribution to represent the signal strength histogram. Can you provide some specific scenarios where it would be a bad approximation?

Your answer:

Gaussian is easy to understand and use and can be entirely described with two parameters. It models the distribution pretty well. However, if the environment is not stable, e.g., perhaps whether a door is open or clsoed might affect the signal strength at a certain location. For this location, if we only apply Gaussian distribution to this location, it would be inaccurate. An alternative would be to use a richer model e.g. a Gaussian Mixture Model or a bimodal distribution

Online phase: find the best match

Make a new version of your `predict()` function above to use this probabilistic technique. In particular, you will need to compute the likelihood that a sample (e.g. the mean of signal strength from AP_1) is actually drawn from the offline-computed distribution. You can compute the likelihood as follows:

```
from scipy.stats import norm

likelihood_AP1 = norm.pdf(trial_signal_strength_AP1, offline_mu_AP1, offline_sigma_AP1)
likelihood_AP2 = norm.pdf(trial_signal_strength_AP2, offline_mu_AP2, offline_sigma_AP2)
likelihood_AP3 = norm.pdf(trial_signal_strength_AP3, offline_mu_AP3, offline_sigma_AP3)
likelihood = likelihood_AP1 * likelihood_AP2 * likelihood_AP3
```

As the Bayes equation shows above, you can simply multiply these probabilities together. An alternative approach, which is numerically more stable, is to instead take the log-likelihood. By taking logs of both sides of the equation, the multiplication turns into an addition, and you can simply add the log-likelihood probabilities from each AP to obtain the overall log-likelihood.

```
from scipy.stats import norm

log_likelihood_AP1 = norm.logpdf(trial_signal_strength_AP1, offline_mu_AP1, offline_sigma_AP1)
log_likelihood_AP2 = norm.logpdf(trial_signal_strength_AP2, offline_mu_AP2, offline_sigma_AP2)
log_likelihood_AP3 = norm.logpdf(trial_signal_strength_AP3, offline_mu_AP3, offline_sigma_AP3)
log_likelihood = log_likelihood_AP1 + log_likelihood_AP2 + log_likelihood_AP3
```

Q: Implement the probabilistic approach to `predict()` and compare its performance to the methods based simply on the mean or the centroid.

Your answer:

> See code below.

Advanced question (optional): Can you plot the error cumulative density function for each of your predict methods?

Your answer:

> See the cells below.

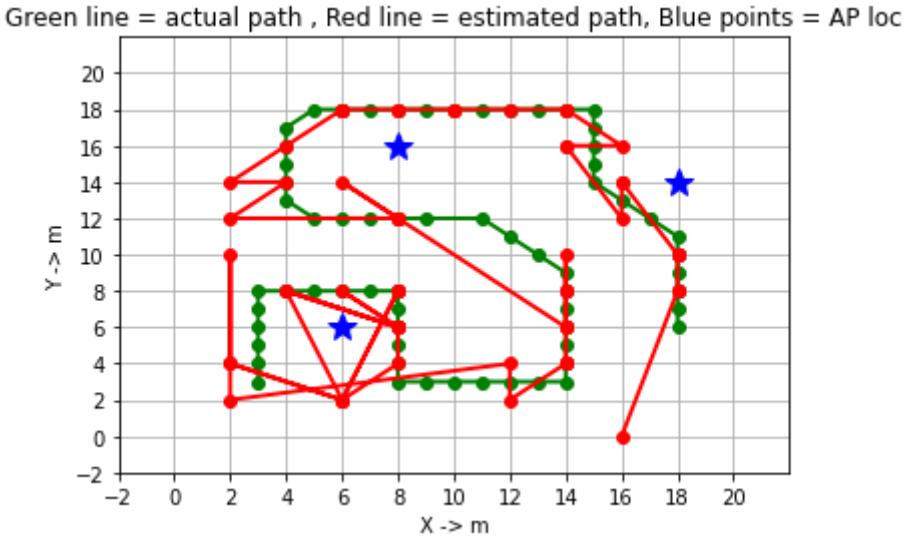
Q: (optional) How well do your methods work for the other dataset, with 5 APs?

This part is left for you to try.

In [28]:

```
#your code here

#Note: this is a different implementation of predict, which returns all the ground truth and predictions as to N*2 arrays.
from scipy.stats import norm
def predict():

    #print "running Horus"

    predicted_loc = np.zeros((len(testData), 2))
    actual_loc = testData[:, 1:3]

    for i in range(0, len(testData)):

        # Keep track of the likelihood being in each location
        loglik = np.zeros(len(normTrainData))

        # Go through each location in the wifi_db
        for j in range(0, len(normTrainData)):

            # Take account all access points
            for k in range(0, NUMBER_OF_APS):
                # Get the RSS data at the unknown location
                ind_start = int(((MAX_SAMPLES - 10) * k) + 3)
                ind_end = int((k + 1) * (MAX_SAMPLES - 10) + 3)
                rssdat = testData[i, ind_start:ind_end]

                # Calculate the mean
                test_point = np.mean(rssdat)

                # Calculate the loglikelihood
                mu = normTrainData[j, (k * 2) + 3]
                sigma = normTrainData[j, (k * 2) + 4]
                loglik[j] += norm.logpdf(test_point, mu, sigma)

        # Find the most likely location
        ind = np.argmax(loglik)
        predicted_loc[i, 0] = normTrainData[ind, 1]
        predicted_loc[i, 1] = normTrainData[ind, 2]

    #print "done."
    return actual_loc, predicted_loc

(actual_loc,predicted_loc)=predict()
plot_path(actual_loc, predicted_loc, 3)
```

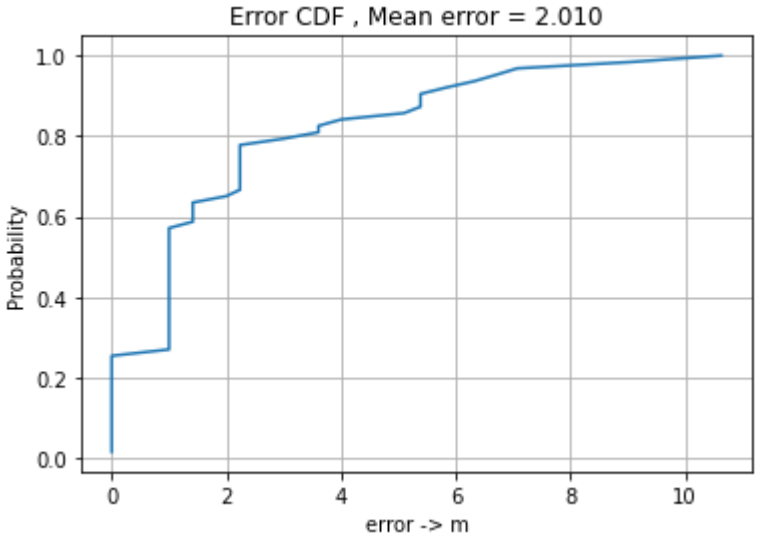
-> Visualizing estimated trajectory...

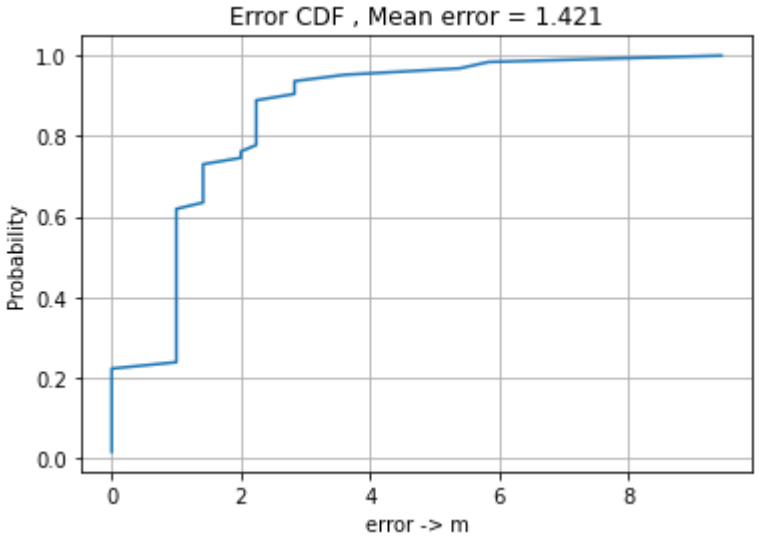


done

In [38]:

```
from statsmodels.distributions.empirical_distribution import ECDF
# Calculate and plot the localization error
def plot_error(actual_loc, predicted_loc):
    """
    :param actual_loc: Actual locations
    :param predicted_loc: Estimated Locations
    :return: null
    """

    #print "-> Calculating localization error..."

    err = np.zeros((1, len(actual_loc)))
    for i in range(0, len(actual_loc)):
        err[0, i] = np.sqrt((actual_loc[i, 0] - predicted_loc[i, 0]) ** 2 +
                            (actual_loc[i, 1] - predicted_loc[i, 1]) ** 2)

    mean_err = np.mean(err)

    ecdf = ECDF(err[0, :])
    plt.figure()
    plt.plot(ecdf.x, ecdf.y)
    plt.grid(True)
    plt.xlabel('error -> m')
    plt.ylabel('Probability')
    plt.title('Error CDF , Mean error = %.3f' % mean_err)
    plt.show(block=False)

    #print "done\n"
    return

plot_error(actual_loc,predicted_loc)

#previous method
plot_error(actual_loc1,predicted_loc1)
```





Q: (optional) Compare and contrast the techniques you have developed today. Consider aspects such as computational cost and factors such as how best to acquire and store the map. How can you update the map? Can you build a map without any surveying at all? Where should the computation happen? What is the impact of device specific variation e.g. different wireless network cards?

Your answer:

> Using the mean is simple and straightforward, but not as accurate as modelling using a Gaussian. If we were doing WiFi based localization on a resource constrained device, fitting a Gaussian to a dataset can be quite expensive. Using the mean/median/interquartile range etc can serve as a simple approximation.

> Storing the map in terms of its distribution (e.g. mean and standard deviation) is cheap, as regardless of the number of measurements, only two parameters are required per location. However, if reality does not align well with the assumptions, e.g. the distribution is long-tailed, bimodal, or even multi-modal, then the Gaussian fit will be a poor approximation.

> To build a map without any surveying is hard, but not impossible. It is called WiFi SLAM (simultaneous localization and mapping). It can be augmented with other information such as from inertial measurement units, or using building floor plans. It is based on the assumption that people do not jump from one location to another location, but can only move to an adjacent location.

> Ideally, the computation should happen on the edge/IoT device, but sometimes it is more accurate to use the cloud, as it can have a richer model. Using a cloud model also allows the model to more easily use updated data from multiple users to build continually evolving maps.

> Due to different sensing abilities, different wireless network cards may perform differently. Thus, there needs to be a per-device calibration phase as well.

In [ ]: