Intro to GPS

In this lab, you will learn how GPS signals are used to estimate the location of an object. GPS satellites broadcast several different signals. These signals contain a very accurate measurement of the satellite's time, as well as the satellite's position, velocity, etc. GPS receivers make use of the fact that light propagates at a known speed, so the receiver can compute distance from the satellite by measuring how long it takes the GPS signal to propagate from the satellite to the receiver. This requires very accurate time measurements—light in free space travels 300m in a microsecond, so small timing errors result in huge distance errors.

The first section of the lab will be to determine how a GPS chip actually goes about receiving and decoding signals. We will step through a subset of problems that must be combatted to successfully send signals from a satellite to your GPS chip. The second portion of the lab will then explore how a GPS chip can use the data it receives to determine its location, assuming the raw, received signals were acquired and decoded. We end with a little open-ended challenge for you. Hope you have fun!

Part I: Data Acquisition

How does a GPS chip decode received signals?

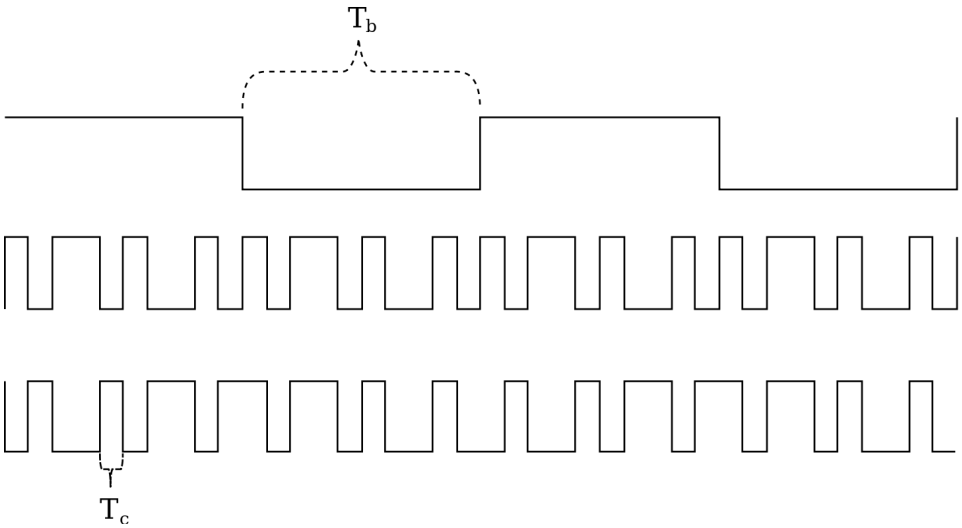
When you a turn a GPS on, it immediately begins to listen for satellite signals. The satellites are continuously transmitting data, and the GPS chip is expected to receive this signal and make sense of it.

The first signal a GPS receiver attempts to find is the Coarse/Acquisition signal. This contains a 1500 bit chunk of data called the "almanac." It contains information and status concerning all the satellites (locations and status) agreed upon by all satellites and is valid for approximately 180 days. This signal is sent at a very low data rate and is intended only to give the receiver a rough idea of the time/location before moving on to the higher data rate, more precise signals. In this lab, we will only focus on the almanac being modulated over the C/A signal.

A simplified version of the C/A signal is depicted and described below.

Data bits (data signals) from each satellite is transmitted at 50 bits / second. This slow data signal is xor'd with a much faster pseudorandom bit sequence (pseudorandom noise, PRN) that repeats every millisecond (1023 samples). Each satellite transmits with a unique PRN that will not correlate with any other satellite's code (the codes orthogonal to one another, and will "cancel each other out" when xor'd together). This is a form of "Code Division Multiple Access," (CDMA) where multiple transmitters can send messages over a single channel without risk of collision. As long as each PRN is orthogonal to the rest, each data signal can be independently recovered.

See http://en.wikipedia.org/wiki/GPS_signals, http://en.wikipedia.org/wiki/Code_division_multiple_access#Steps_in_CDMA_modulation, and come talk to us if you're interested in learning more about this!



The following code allows you to simulate an idealized GPS receiver with some plausible parameters.

The pseudorandom code is generated by a linear feedback shift register (see http://en.wikipedia.org/wiki/Linear_feedback_shift_register).

The function `transmit_to_earth(signal)` simulates what the signal might look like by the time it reaches your receiver. Skim through the code, but don't worry if you don't understand all of it.

In [ ]:
```
import numpy as np
import scipy.signal

# linear feedback shift register, default for taps is that used by GPS C/A Sig
nal
def lfsr(n, starting_state=(1 << 10) - 1, taps=[3,10]):
    state = starting_state
    for i in range(n):
        yield state
        state = ((state << 1) & ((1 << max(taps))-1)) + reduce(lambda x,y: x^y
, map(lambda x: state & (1<<(x-1)) != 0, taps), False)

# coarse/acquisition code generation. reset is all 1's state. This is data tha
t is modulated by a pseudorandom bit sequence.
def ca(starting_state=(1 << 10) -1):
    return np.fromiter( ( 2*(i >> 9)-1 for i in lfsr(1023, starting_state=star
ting_state)), dtype=np.float)
ca_canonical=ca()

offset = int(np.random.uniform(0, ca_canonical.size))

# adapted from http://common.globalstar.com/doc/axonn/GPS-L1-Link-Budget.pdf
def transmit_to_earth(signal, temp=290, offset=offset, bw=2e6, SNR_boost=0, NF
=0):
    signal = np.roll(signal, -offset) # add a random phase
```

```
    elevation = 2.5e7 #m, approximately over the horizon
    antenna_gain = 13. #dBi
    power = 46.5 #dBm
    c_lambda = .1904 #m
    temp = 290 #K
    thermal_noise = 10*np.log10(1.38e-23 * temp) + 30 #kT in dBm
    # Carrier to Noise ratio in dB
    CbyN0 = power + antenna_gain - 20 * np.log10(4*np.pi * elevation / c_lambd
a) - thermal_noise
    SNR = CbyN0 - 10*log10(bw) + SNR_boost - NF
    #print SNR
    return signal + 10**(-SNR/20.) * np.random.normal(size=signal.size)
```

Problem 1: Signal Strength

GPS satellites have limited power and need to spread their signals over the entire surface of the earth, so the signals are very weak by the time they get to the GPS receiver. As a result, thermal noise and noise from other sources will be large compared to the signal. The code below plots the received signal in the time domain. There is a slider on the bottom that you can move around to boost the signal to noise ratio (SNR) in dB. The SNR measures exactly what you would expect it to: what is the ratio of actual signal to noise in the received message. As an analogy, think of yourself talking to a friend in a loud, crowded room. In order for your friend to hear you, you most likely have to speak very loudly to be heard over the background noise. Imagine what you would have to do to be heard from across the room! In a similar fashion, as you increase your SNR, you are more likely to get your signal across successfully. However, just as you need to exert more energy to speak louder, this also requires more power on the satellite's end.

In []:
```
%pylab
# from nbviewer.ipython.org
from matplotlib.widgets import Slider

fig, ax = plt.subplots()
fig.subplots_adjust(bottom=0.2, left=0.1)

x = np.linspace(0, 1023, len(ca_canonical))
line, = plt.plot(x, [0.]*len(x))
ax.set_xlim([0, 1023])
ax.set_ylim([-5,5])
def on_change(val):
    line.set_ydata(transmit_to_earth(ca_canonical, SNR_boost=val))
on_change(0)

slider_ax = plt.axes([0.1, 0.1, 0.8, 0.02])
slider = Slider(slider_ax, "Noise Figure", 0, 50, valinit=0, color="#AAAAAA")
slider.on_changed(on_change)
```

```
print "Look for offset = " + str(offset)
```

Q1: By approximately how much do you need to boost the SNR for the received signal to look "clean"? What was the original SNR of the signal?

This is subjective, but there should be some SNR where it stops looking like garbage and starts looking like the signal. Is it reasonable to ask a satellite to use this much more power?

A1. Your answer here

Problems 2 and 3: Signal Starting Point and Decoding

The next problem is that when the GPS first starts up and hears a signal, it doesn't know when the data starts. Note that `transmit_to_earth()` "rolls" the input by a randomly generated offset in order to simulate the fact that a GPS receiver doesn't know where the bits start and end.

Once a receiver knows this time offset, it knows the time to \(<1\)ms (unfortunately, light goes really far in a millisecond). The next step is for the receiver to take each group of 1023 samples and figure out if they correspond to a `1` bit or a `0` bit. A GPS receiver needs to do all of these tasks despite the signal being weaker than the noise!

The basic tool for achieving these tasks is the matched filter (http://en.wikipedia.org/wiki/Matched_filter). Matched filters perform a correlation on an input signal with an expected reference signal. A matched filter performs a convolution with the time-reversed, conjugated signal, which essentially amounts to a sliding dot product (remember, convolution time-reverses and conjugates the signal, generally, so if you time-reverse and conjugate in the first place, then the operation becomes a simple sliding dot product). The idea is that this slidiing dot product will in general be small, until the two signals precisely align, where you will see a spike. If the two signals are aligned, \(\sum_{i=0}^N r_i * (b * r_i) = Nb\) (recall \(r_i\) is \(\pm 1\)). If the two signals are not aligned, because we have chosen our sequence to look random, we say \(r_i\) is approximately independent from \(r_k\) if \(k\ne i\), so \(\sum_{i=0}^N r_i (b * r_{i+k})\) has expectation approximately 0.

Q2. In the space below, implement a function that performs matched filtering, i.e. performs a correlation on `signal` and `reference`. It should be able to handle `signal` and `reference` being different sizes.

In [4]:
```
def matched_filter(signal, reference=ca_canonical):
    #Your code here
    pass
```

If you have correctly implemented matched_filter, the below code should plot the result of matched filtering your noisy signal. The slider on the bottom boosts (or reduces) the SNR. If you boost the SNR, you should see a peak at offset (your particular random offset is printed by the above code). Cool stuff!

In [ ]:
```
%pylab
# from nbviewer.ipython.org
```

```
from matplotlib.widgets import Slider

fig, ax = plt.subplots()
fig.subplots_adjust(bottom=0.2, left=0.1)
x = np.linspace(0, 1023, len(ca_canonical))
line, = plt.plot(x, [0.]*len(x))
ax.set_xlim([0, 1023]), ax.set_ylim([0,80])
def on_change(val):
    line.set_ydata(10*np.log10(matched_filter(transmit_to_earth(ca_canonical,
SNR_boost=val))**2))
on_change(0)

slider_ax = plt.axes([0.1, 0.1, 0.8, 0.02])
slider = Slider(slider_ax, "SNR Boost", -10, 10, valinit=0, color="#AAAAAA")
slider.on_changed(on_change)
print "Look for offset = " + str(offset)
```

Q3 When the matched filter is aligned with the input, the output is $X=Nb + \sum_{i=0}^N v_i$, where $v_i\sim N(0,\sigma^2)$ is some additional noise. For our value of $N=1023$, what is the variance of our estimator $\hat{b}=\frac{1}{N}X$? What is the SNR of $\hat{b}$? How much bigger is it than our original SNR? (Recall $b\in \{ -1, 1\}$). Based on Q1, is this enough to make our signal look clean?

Q4 Your answer here

For the higher datarate GPS signals that give more precise timing information, \(N\) is smaller and the noise averaging takes longer. To average out enough noise to get a good lock, GPS receivers need to correlate for a long time. This is the primary reason it can take a long time for your GPS to figure out where you are.

Problem 4: Noise Figure

So far, we have assumed that our receiver detects the signal perfectly. In reality, no receiver is perfect and designers must work around many nonidealities. One common problem is noise figure. Before converting a signal from analog to digital, receivers pass the signal through an amplifier to get the weak signal to a high enough level for sampling. Unfortunately, active components like amplifiers add their own noise to the system. The parameter that measures this noise is called noise figure—it measures how much extra noise an active component adds to a signal.

For the purposes of this lab, we can treat \(SNR_{out}=SNR_{in} - NF\), where \(SNR_{in}\) is the SNR of the signal before going through the amplifier, \(NF\) is the noise figure of the amplifier, and \(SNR_{out}\) is the SNR of the signal at the output of the amplifier. All values are in dB.

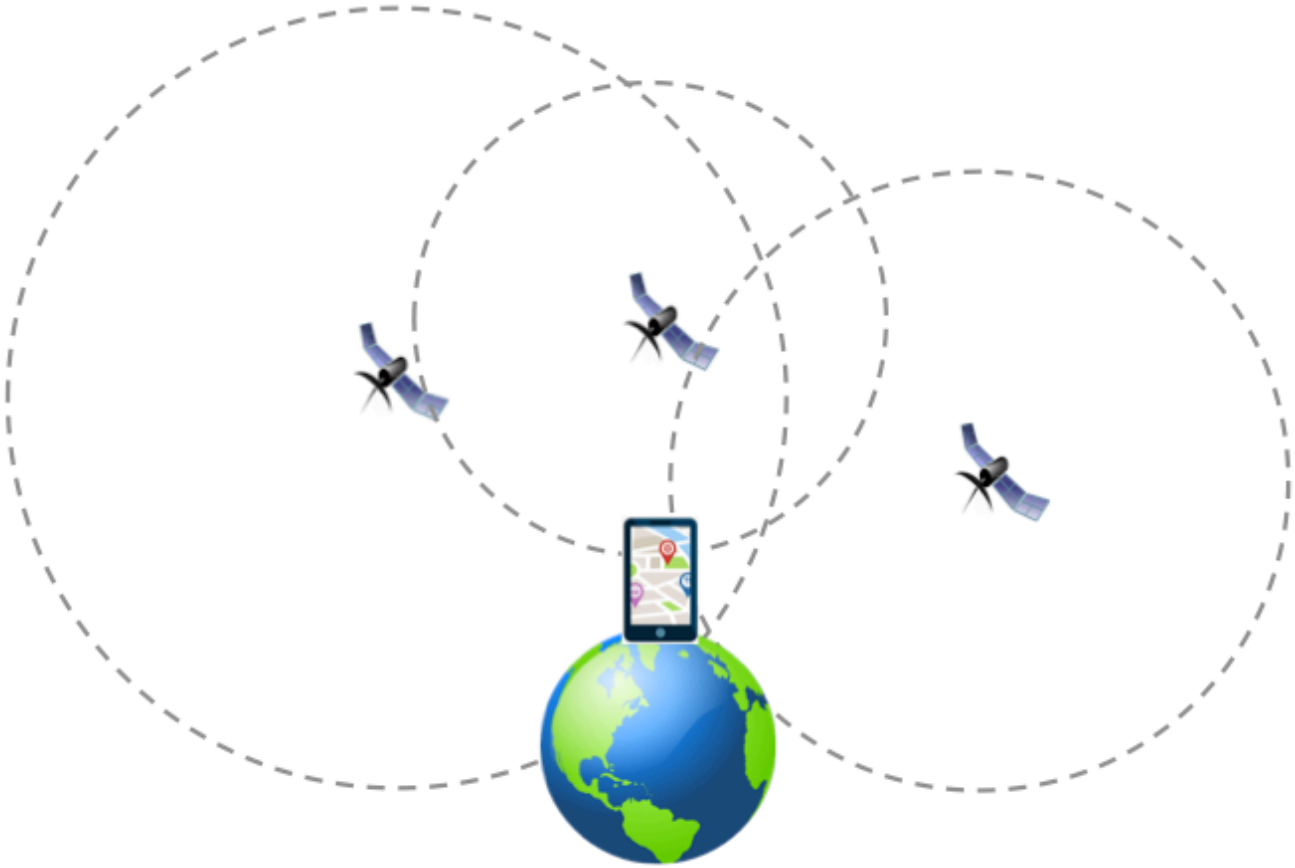
Q4 Write code that generates 100 different random offsets. Use the `noise_figure` and `offset` parameter of `transmit_to_earth` to generate 100 different simulated signals with noise figures of 1dB and 8dB. Use your matched filter code to find the offsets for the two different noise figures. Which noise figure performs better? By how much?

Great! Now you know how GPS signals are transmitted and received :D

Part II: Finding your location

How does a GPS chip determine where it is located

Let's switch gears a bit and explore what a GPS chip does in order to use the obtained data effectively. We'll build up a simple model, and then let you pretend to be the GPS chip.



We first assume that we have obtained the N GPS signals (from the first part), each of which gives a noisy measurement of the distance between the GPS satellite and the object.

The noisy measurements are modeled as follows, where n_i is iid Gaussian noise with zero mean and variance σ^2. $$D_i = d_i + n_i$$

In the above equation, d_i is the actual distance to the i-th satellite, and D_i is the reported distance, which is corrupted by additive noise n_i. This additive white gaussian noise (AWGN) channel model is actually very common in information theory, and can be analyzed just like how we analyzed the BEC and BSC earlier in the course! It is a pretty good model for satellite communication links as you don't have to deal with shadowing, multipath, excessive interference, etc. (come talk to one of us if you're interested in learning more about this stuff/what it means!)

For simplicity, let's visualize the entire space as a 2D plane. Assume that all GPS satellites and the object to be located (GPS chip) are on the plane. Denote the position of the unknown object as (x, y), and let (x_i, y_i) be the position of the ith GPS satellite.

Q5. Consider the case where $\sigma=0$, i.e., noiseless distance measures are given. In the 2D case, what is the minimum N necessary to estimate the unknown position exactly?

(hint: This isn't supposed to be too theorectical. Think geometry!)

A5 Your answer here

Q6. What about in the 3D case?

A6 Your answer here

Q7. Going back to the 2D case, find the MLE of (x, y) given $\{(x_i,y_i)\}$ and $\{D_i\}$.

Hint 1: Leave your answer in the form of an expression to be maximized (as in, something proportional to the likelihood function)

Hint 2: Use the distance formula. What is the relationship betweek d_i and (x_i,y_i)?

A7 Your answer here

Q8. Open ended question! For the following sets of measurements, find the unknown location. This is real GPS data, so we'll see how close you get to the true value.

You can find a way to use the estimator you derived above, modify it as you wish, or do something completely different. We will rank the submitted answers and give you bonus points depending on your rank. Print your guess on a single line underneath any other output you have.

In [ ]:
```
# unknown_position = (?, ?)
sensor_position = [(-40.694743669342756, -20.226418811183823), (-32.759267680406
58, 76.86105781285312), (82.72920374310364, 92.09272797391658), (-76.1064053
244318, 161.00206304086052), (80.28971616305012, -170.69523789882172), (155.71
668299915098, 15.181435053970823), (7.939009827243231, -34.2712304358226), (-1
78.40905705845432, 60.52718995632699), (90.29370103086475, -129.9134744252411)
, (10.740021155140816, 86.14218126967965), (35.018649652481606, 172.7138366992
172), (-36.64404003662175, 141.39719318173255), (-109.66699375945394, -97.9280
971937537), (-40.919777210810395, 12.708488516937662), (0.9443591850516908, -1
31.77559700648598), (83.29915631942923, 112.0737057262707), (168.0460198752878
, -143.91703484549907), (-83.40828690260733, -50.594711084070624), (137.235330
59019437, 143.33931382080246), (61.95943087474895, 56.80260189453879), (-155.0
7668103825543, -89.41747331798778), (30.742636682015466, -79.98246328953933),
(37.7054760371075, -61.73535570658275), (-56.580649473737594, 84.5559569857084
6), (4.886318816386646, -118.04469383560077), (155.8085044558207, -43.19725966
261523), (76.27813460373156, -48.9409986676602), (57.57934263732175, 93.822664
```

11339654), (63.03939105522883, -21.256635168542406), (135.62514808957295, -44.36378671834361), (-123.70977670663869, -81.36386500586175), (-96.0285390219639 1, 32.84351069374866), (130.11218359151763, -69.15682783073973), (95.210535588 07434, -45.96944653330596), (-149.67971565049027, 74.25995945394389), (-1.0325 635312033077, -80.71310744261156), (-81.06063338667441, 106.85176287971301), (135.21545265818543, -175.94066300476234), (51.605549769896236, 162.03595676220 97), (1.7368707349554002, -14.430514347965829), (88.61860883947605, -5.3182578 76386672), (12.901706676280437, -119.28347775433457), (10.994810487999956, -14 .853348237002816), (-169.56157078961064, 75.77204164605116), (108.436631670973 92, -17.07851326442877), (-33.691584722923935, 69.55771949272305), (-78.625622 26118584, -31.821651323450606), (29.59029173040532, 153.9912698190567), (-110. 68145481884578, 138.15755196857438), (96.36118809899352, -11.30354393211106), (-59.13761990283959, 150.20374185296066), (36.866419534781265, 131.57711926255 803), (-155.69587986503072, 151.46336017398926), (-168.75441324844553, 83.4988 3121116615), (-60.64049658364907, -93.22467266072562), (-125.87748072220982, - 86.31398113526421), (-65.30679518448372, -75.40188398011682), (-86.80847499725 564, -126.04730512846959), (131.47614522645765, 95.57522114439986), (-68.44877 530479526, 18.060719124840926), (38.526163654634, -114.69348325002177), (-31.8 28299342061136, 71.2618610818466), (132.19970032227386, -111.4974935870704), (55.53705008650195, -163.47740141578555), (-112.84973728183768, -48.81386494696 673), (-110.59300868711388, -65.45308897768105), (26.76040985023888, -161.1741 9631227605), (-141.14270746482651, -62.4674499192967), (43.195471675489834, -1 09.70440740965005), (-107.98061225426493, -93.45726832183934), (55.28402734900 8845, -22.373209570892687), (107.79743223438453, 81.98820216042571), (-175.473 94257921172, -1.5625065712816433), (52.22099974418929, -167.32648464729073), (-159.42861863592685, -120.65920426291595), (27.967733815517676, 173.7908788203 4463), (-79.81108076559832, 12.159862001672161), (-1.083066403959192, 10.14405 4135055987), (18.556794720265735, -14.232424503082077), (133.70293010631914, 5 8.859464423989095), (-6.027114910694675, 83.5750817482432), (139.8603494911298 5, -165.9387111702894), (-108.38010180568251, -43.43786921956776), (53.1356647 91481, -132.62752111271098), (-157.9670089193256, 164.29626275241714), (41.745 06994237683, -134.31331339187747), (-84.75344368921516, -44.20483413406672), (81.84797984571499, 25.05981797780347), (47.98082058394565, 11.746190092001832) , (56.54218492633772, -142.0718864968182), (-138.90701288445976, -157.76013236 43112), (8.710873072113072, -124.55938916794305), (148.33316094229656, 64.8535 0570573632), (133.14496791855365, -2.0638101399426567), (-48.285588716728306, 130.01710621200243), (-2.5790741780163184, -117.46994923047784), (13.798742282 537301, 139.9808900002956), (136.5458026341664, 142.90550786586383), (-141.484 23564741253, 51.526862136327594), (55.552509852399005, -55.019705619247404)]
measured_dist = [130.90641473610214, 209.29824954319642, 217.31947498448309, 3 04.344966403917, 67.231487401346968, 180.23072235777173, 94.167831905947182, 2 84.70263830670302, 47.605747896019892, 211.57071581692631, 293.02914282518299, 273.70053111447095, 148.64471889074713, 154.02874947327001, 36.364991813309985 , 239.25892207401549, 131.20905739684972, 135.59924400192486, 287.275782307546 99, 176.8582317310925, 197.37459332532916, 46.986073529556052, 63.728127096772

```
376, 226.73665833127751, 36.519754372951041, 140.26925185798973, 81.4769066684
94507, 219.21282385494803, 105.267376996856, 126.58444868340739, 168.615417307
05429, 204.14847155064396, 109.47797005492703, 93.404669303189394, 268.9363801
182003, 52.760578723790481, 257.03225424062401, 111.98125668158696, 286.980241
57644699, 117.33929273892588, 125.81354568760531, 24.060612768806209, 110.1794
2055793434, 291.01115246809343, 124.02559574989046, 206.06684022082845, 153.13
567660980365, 272.03979344597656, 296.49375976704795, 124.90090658390021, 292.
37026606357233, 251.37757305817749, 335.61980232663939, 292.38640011695435, 98
.571650488574519, 170.48910477029588, 105.18136159110239, 127.61839334346715,
238.55494981198035, 175.72465623397665, 8.857630482788089, 203.70277832427414,
95.234020237909206, 45.896248056609103, 168.78348939760147, 157.23174334515585
, 39.458356929550177, 185.53941391408924, 17.05753212865697, 145.4466378130659
3, 101.40510630733674, 219.85587923103691, 244.41566521878821, 47.231082292323
912, 194.51369916905452, 297.481153486452, 177.85316280627691, 136.71074013904
112, 109.13746413244726, 199.19639342698414, 209.21782664085654, 107.729419748
80837, 166.36338526660887, 22.385533472503166, 344.92902439223741, 8.065277842
759528, 144.07893090886935, 154.41730582061345, 137.25653814647268, 27.0404483
43013878, 178.10838922002972, 33.099559359942745, 216.91639488004489, 154.5285
129873823, 263.96787232485269, 35.350585292584732, 266.41961938357417, 287.158
16324601258, 249.25743694835796, 68.341683239707848]

def YOUR_GPS_ALGORITHM( sensor_position, measured_dist ):
    # BLAH BLAH

    return you_are_here
```