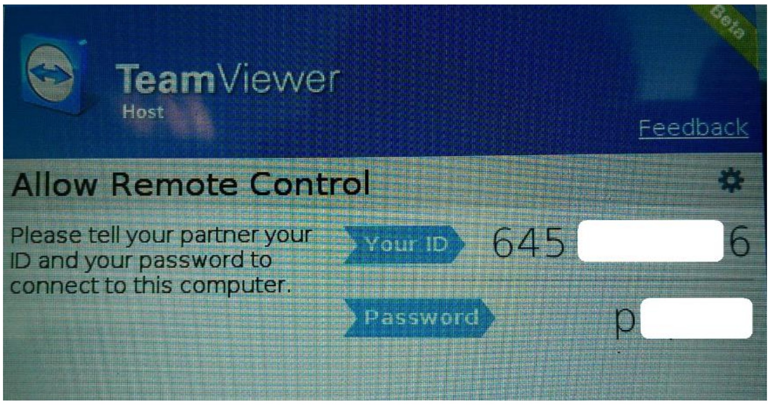
Remotely control Raspberry Pi (teamviewer)

On Pi:

```
wget http://download.teamviewer.com/download/linux/version_11x/teamviewer-host_armhf.deb
sudo apt-get -f install
sudo dpkg -i teamviewer-host_armhf.deb
sudo apt-get install -f
```



Allow Remote Control

Please tell your partner your ID and your password to connect to this computer.

Your ID 645[redacted]6

Password p[redacted]

On your laptop:

Install teamviewer on their website

Enter your ID and Password on your laptop’s teamviewer, then you can use your laptop to remotely control your Pi.

Import python libraries on Raspberry Pi

```
sudo pip install python boto ## change boto to the libraries you want
```

Raspberry Pi HAT data collection

```
sense = SenseHat()
for i in range(datanum): ## change to your data number
        message = {}
        device = {}
        device['cpuTemperature'] = os.popen('vcgencmd measure_temp').readline().replace("temp=","").replace("'C\n","")
        environment = {}
        temp = {}
        temp['basedOnHumidity'] = sense.get_temperature_from_humidity()
        temp['basedOnPressure'] = sense.get_temperature_from_pressure()
        environment['humidity'] = sense.get_humidity()
        environment['pressure'] = sense.get_pressure()
        environment['temperature'] = temp
        message['device'] = device
        message['environment'] = environment
        jsonData = json.dumps(message)
        #sleep(0.5)
```

Communication between Pis (Bluetooth)

1. Get two machines running Raspbian. One will be the server and the other will be the client.
2. On both machines, install python-bluez. sudo apt-get install python-bluez
3. Find out the server's Bluetooth address (e.g. using "hcitool dev")
4. Pair the client and server using Bluetooth address

For client Pi:

```
bd_addr = "B8:27:EB:3F:84:11"   # here need to modify. use hciconfig to get address
port = 1
sock=bluetooth.BluetoothSocket( bluetooth.RFCOMM )
sock.connect((bd_addr, port))
sock.send(cPickle.dumps(data)) ## because the matrix cannot be transmitted directly, you need to use cPickle.dumps()
sock.close()
```

For server Pi:

```
server_sock=bluetooth.BluetoothSocket( bluetooth.RFCOMM )
port = 1
server_sock.bind(("",port))
server_sock.listen(1)
client_sock,address = server_sock.accept()
data = client_sock.recv(1)
data_all = pickle.loads(''.join(total_data)) ## pickle.loads() can be used to restore your data type
```

Connect to AWS dynamoDB:

```
conn = boto.dynamodb.connect_to_region(
    'us-east-2',## the region of your account
    aws_access_key_id='',
    aws_secret_access_key='')
```

You can find your access key and secret access key on the following website

https://aws.amazon.com/blogs/security/wheres-my-secret-access-key/

1. Creat table:

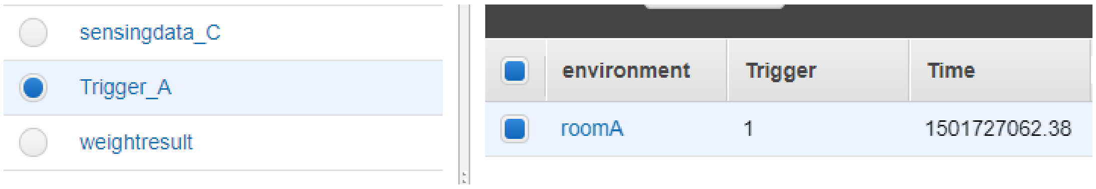
```
message_table_schema = conn.create_schema(
    hash_key_name = ‘environment’,
    hash_key_proto_value =str,
    range_key_name = ‘Trigger_A’,
    range_key_proto_value =str, (str can be changed to int…..)
    )
table = conn.create_table(
    name=’ Trigger_A’,
    schema= message_table_schema,
    read_units=1, # if your want to transmit a large number of data, making this
number larger will be better
    write_units=1, # if your want to transmit a large number of data, making this
number larger will be better
    )
```

2. Write new data to dynamoDB

```
table = conn.get_table(‘Trigger_A’)
item_data = {
    ‘Time’: 1501727062.38, ##you can add anything you want
    }
item = table.new_table(
    hash_key = ‘roomA’
    range_key = ‘1’
    attrs = item_data
    )
Item.put()
```

3. Get item from dynamodb

```
table = conn.get_table('Trigger_A')
item = table.get_item(
        hash_key='roomA',
        range_key='1',
    )
print item[‘Time’]
>> 1501727062.38
```



■	environment	Trigger	Time
■	roomA	1	1501727062.38

Or you can also update your dynamodb table:

```
item[‘Time’] = 123
print item[‘Time’]
>>123
```

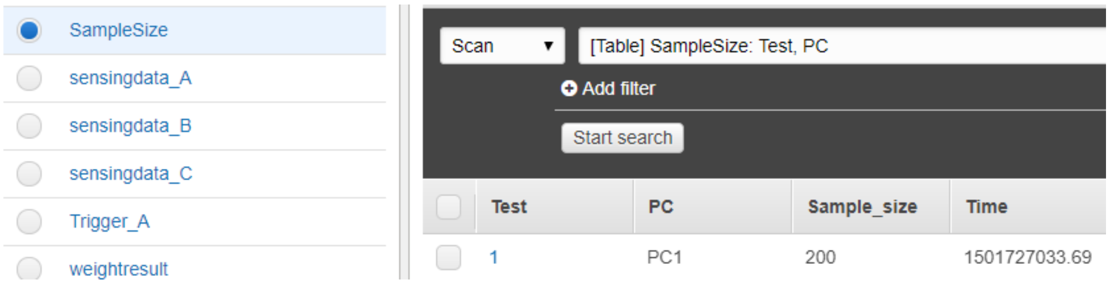
Lambda

If you want to import another python libraries, follow the step on the following websit: https://github.com/vitolimandibhrata/aws-lambda-numpy

If you only want to use numpy, you can just download all documents on the wedsite.

1. Get item from dynamodb

```
dynamo = boto3.resource('dynamodb')
table_S = dynamo.Table('SampleSize')
db_response_S = table_S.scan()
item_S = tuple(db_response_S['Items'])
###because the data stored in the table would be like this: {u'Count': 1, u'Items': [{u'Test': u'1', u'Sample_size': u'200', u'Time': Decimal('1502218588.4'), u'PC': u'PC1'}], you need to use tuple() to get each item in ‘Items’
datanum = int(item_S[0]['Sample_size'])
print datanum
>>200
```

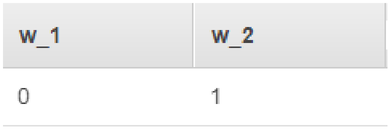


[Table] SampleSize: Test, PC

□	Test	PC	Sample_size	Time
□	1	PC1	200	1501727033.69

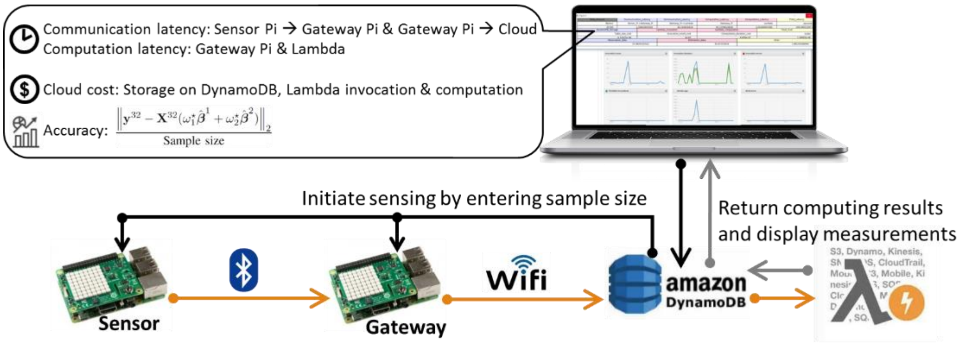
2. Store data to dynamodb:

```
table = dynamo.Table('weightresult')
item = table.put_item(
            Item = {
                  'w_1':0,
                  'w_2':1,## you can add some item you want
                  }
      )
```



w_1	w_2
0	1

In my system:



Communication latency: Sensor Pi → Gateway Pi & Gateway Pi → Cloud
Computation latency: Gateway Pi & Lambda
Cloud cost: Storage on DynamoDB, Lambda invocation & computation
Accuracy: $\frac{\left\|\mathbf{y}^{32} - \mathbf{X}^{32}(\omega_1^{*}\hat{\beta}^{1} + \omega_2^{*}\hat{\beta}^{2})\right\|_2}{\text{Sample size}}$
Initiate sensing by entering sample size
Return computing results and display measurements
Sensor
Gateway
Wifi
amazon DynamoDB

Required python libraries:

On sensor Pi:

import bluetooth

import os

import json

import cPickle

import numpy as np

from time import sleep

from sense_hat import SenseHat

import time

import matplotlib.pylab as plt

import boto

import boto.dynamodb

On gateway Pi:

import bluetooth

from time import sleep

import csv

import json

import cPickle

import pickle

```
import numpy as np
import random
import os
import socket
import ssl
import json
from time import sleep
from sense_hat import SenseHat
#from boto.s3.connection import S3Connection
import boto
#from boto.s3.key import Key
from socket import gaierror
from ConfigParser import NoOptionError
import time
import boto.dynamodb
import select
import matplotlib.pylab as plt
```

Code for sensor Pi:

```
sense = SenseHat() # must need this to drive your HAT
```

LED:

```
X = (0, 255, 0) # Green
O = (0, 0, 0) # Black
creeper_pixels_1 = [
        O, O, O, X, X, O, O, O,
        O, O, X, O, O, X, O, O,
        O, X, O, O, O, O, X, O,
        X, O, O, O, O, O, O, X,
        X, O, O, O, O, O, O, X,
        O, X, O, O, O, O, X, O,
        O, O, X, O, O, X, O, O,
        O, O, O, X, X, O, O, O
]

sense.set_pixels(creeper_pixels_1) # show LED
```

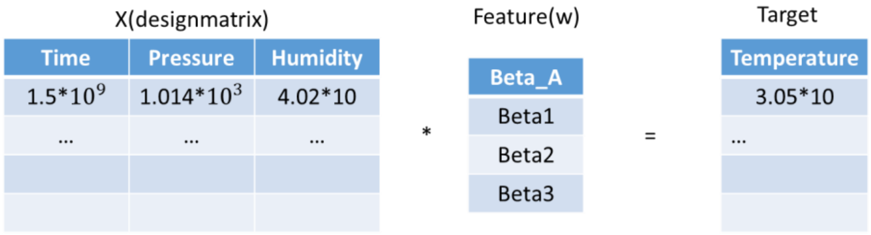
Get sample size from dynamodb:

```
conn = boto.dynamodb.connect_to_region(
    'us-east-2',
    aws_access_key_id='', # your key
    aws_secret_access_key='') # your key

table = conn.get_table('SampleSize')
### see if the table is updated
while True:
    item = table.get_item(
                # Our hash key is 'forum'
                hash_key='', #your key
                # Our range key is 'subject'
                range_key='', #your key
                # This has the
                #attrs=item_data
                )

    if old_size_time != item['Time']:
        old_size_time = item['Time']
        break
###get new sample size from dynamodb
datanum = int(item['Sample_size'])/2 # separate sample size to gateway &sensor
equally
feature = 3
data_all = np.zeros((datanum,feature+1)) #[timestamp pressure humidity
temperature]
###use sense HAT to sense data
for i in range(datanum):
        #led
        sense.set_pixels(creeper_pixels_1)
        #led
        message = {}
        device = {}
        device['cpuTemperature'] = os.popen('vcgencmd
measure_temp').readline().replace("temp=","").replace("'C\n","")

        environment = {}
        temp = {}
        temp['basedOnHumidity'] = sense.get_temperature_from_humidity()
        temp['basedOnPressure'] = sense.get_temperature_from_pressure()
        environment['humidity'] = sense.get_humidity()
        environment['pressure'] = sense.get_pressure()
        environment['temperature'] = temp

        message['device'] = device
        message['environment'] = environment

        jsonData = json.dumps(message)
        tdataTime = time.time() #use this to record timestamp in anywhere if you want

        data_all[i][0] = tdataTime
        data_all[i][1] = sense.get_pressure()
        data_all[i][2] = sense.get_humidity()
        data_all[i][3] = sense.get_temperature_from_humidity()

###
bd_addr = "XX:XX:XX:XX:XX:XX"    # here need to modify. use hciconfig to get address
port = 1
sock=bluetooth.BluetoothSocket( bluetooth.RFCOMM )
sock.connect((bd_addr, port))
sock.send(cPickle.dumps(data_all)) # need to change data type to make the transmission successful.
sock.close()
```

Code for gateway Pi:

Gateway Pi also need to get new sample size from dynamodb, all this part is same as sensor Pi.

```
datanum = int(item['Sample_size'])/2
feature = 3
designmatrix = np.zeros((datanum*2,feature))
Target = np.zeros((datanum*2,1))
#####initial w(beta)#####
w_old = np.zeros((feature,1))
w_new = np.zeros((feature,1))
```



X(designmatrix)

Time	Pressure	Humidity
$1.5*10^9$	$1.014*10^3$	4.02*10
…	…	…

*

Feature(w)

Beta_A
Beta1
Beta2
Beta3

=

Target

Temperature
3.05*10
…

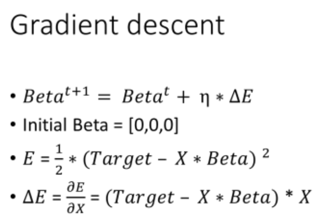
```
E_old = 0
E_new = 0
delta_E = np.zeros((datanum*2,feature))
learning_rate = 0.001
###bluetooth_server###
server_sock=bluetooth.BluetoothSocket( bluetooth.RFCOMM )
port = 1
server_sock.bind(("",port))
server_sock.listen(1)
try:
    client_sock,address = server_sock.accept()
    print "Accepted connection from ",address
    tStart_blue_recv = time.time()
    total_data = []
    while True:
        data_1 = client_sock.recv(1)
        if len(data_1) == 0: # if transmission finished, then break
            break
        total_data.append(data_1)
except IOError:
    pass
except KeyboardInterrupt:
    print "Stopping..."
    stop_advertising(server_sock)
    sys.exit()
data_all = pickle.loads(''.join(total_data)) # need to pickle the data because I used cPickle.dumps() on my sensor Pi

client_sock.close()
server_sock.close()
```

The next part is using gateway Pi to collect data, all this part is the same as sensor

Pi.

```
### Gradient descent ###
while True:
        w_old = w_new
        for i in range(datanum*2):
                delta_E[i,:] = delta_E[i,:] + (Target[i][0] -
np.dot(np.matrix(designmatrix[i,:]),np.matrix(w_old)))*designmatrix[i,:]
                ###normalization##
                #delta_E[i,:] = ((delta_E[i,:] - np.min(delta_E[i,:]))/(np.max(delta_E[i,:])-
np.min(delta_E[i,:])))*2-1
                #delta_E[i,:] = delta_E[i,:] - np.min(delta_E[i,:])
                #delta_E[i,:] = delta_E[i,:]/np.max(delta_E[i,:])

        w_new = w_old + learning_rate*np.matrix(delta_E[i,:]/(datanum*2)).T
        print "beta_old = ", w_old
        print "beta_new = ", w_new

        E_old = E_new
        for i in range(datanum*2):
                E_new = E_new + (Target[i][0] -
np.dot(np.matrix(designmatrix[i,:]),np.matrix(w_new)))**2
                E_new = E_new/2

        if E_new > E_old:
                learning_rate = learning_rate/2
```

The following are the formulations of gradient descent:



Gradient descent

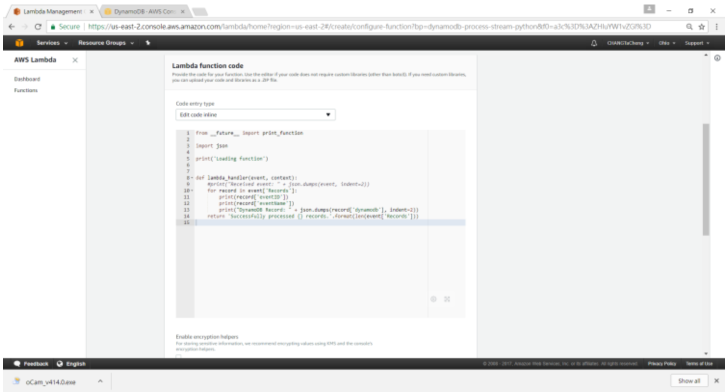
- $Beta^{t+1} = Beta^{t} + \eta * \Delta E$
- Initial Beta = [0,0,0]
- $E = \frac{1}{2} * (Target - X * Beta)^{2}$
- $\Delta E = \frac{\partial E}{\partial X} = (Target - X * Beta) * X$

```
### Connect to dynamodb and do some interaction###
table = conn.get_table('sensingdata_A')
item = table.get_item(
                # Our hash key is 'forum'
                hash_key='roomA',
                # Our range key is 'subject'
```

```
            range_key='sensorA',
            # This has the
            #attrs=item_data
            )

tStart_pi_lambda = time.time()
item['feature_A'] = float(w_new[0][0])
item['feature_B'] = float(w_new[1][0])
item['feature_C'] = float(w_new[2][0])
item['Comm_pi_pi'] = bluetooth_transmission_time
item['Compu_pi'] = Compu_pi_time
item.put()
tEnd_pi_lambda = time.time()
Comm_pi_lambda = tEnd_pi_lambda - tStart_pi_lambda
item['Comm_pi_lambda'] = Comm_pi_lambda
item['Time'] = tEnd_pi_lambda
item.put()
```

On Lambda:

1. Click Services -> IAM -> Roles -> lambda_dynamodb_role -> create role policy -> policy generator -> select -> AWS Service (choose Amazon DynamoDB) -> Actions (all actions(*)) -> Add statement -> Amazon Resource Name (ARN) (you can find this on your dynamodb table -> overview) -> Add statement -> next step
2. Click Services -> lambda -> Function -> Create function -> dynamodb-process-stream-python
3. At configure trigger, set the table which you want to use to trigger lambda (starting position choose latest)
4. At configure function, set you lambda function name.
5. You can directly design your code here but you can only use the default python libraries, or upload the zip which include your code and extra python libraries.



Lambda function code

Provide the code for your function. Use the editor if your code does not require custom libraries (other than boto3). If you need custom libraries, you can upload your code and libraries as a .ZIP file.

Code entry type: Edit code inline

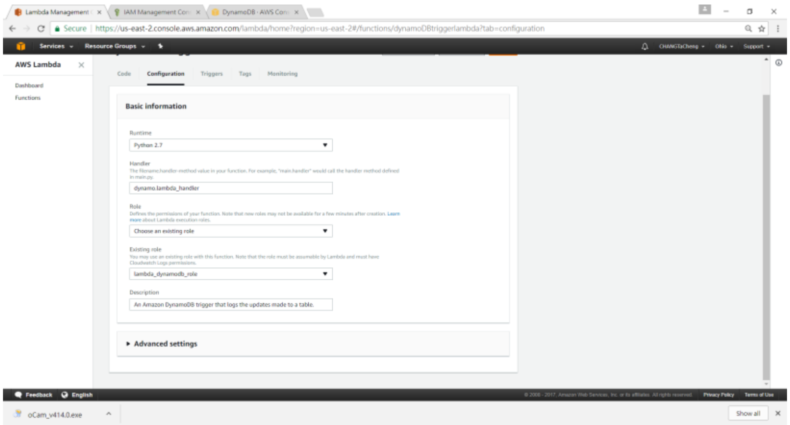
```
from __future__ import print_function

import json

print('Loading function')


def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))
    for record in event['Records']:
        print(record['eventID'])
        print(record['eventName'])
        print("DynamoDB Record: " + json.dumps(record['dynamodb'], indent=2))
    return 'Successfully processed {} records.'.format(len(event['Records']))
```

Enable encryption helpers

Lambda Management
DynamoDB - AWS Con
Secure
https://us-east-2.console.aws.amazon.com/lambda/home?region=us-east-2#/create/configure-function?bp=dynamodb-process-stream-python&f0=a3c%3D%3AZHluYW1vZGI%3D
Services
Resource Groups
CHANGTaCheng
Ohio
Support
AWS Lambda
Dashboard
Functions
Feedback
English
oCam_v414.0.exe
Show all

Code for lambda:

First, click your lambda function -> configuration -> check the name of handler



Basic information

Runtime: Python 2.7

Handler: The filename.handler-method value in your function. For example, "main.handler" would call the handler method defined in main.py.

dynamo.lambda_handler

Role: Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. Learn more about Lambda execution roles.

Choose an existing role

Existing role: You may use an existing role with this function. Note that the role must be assumable by Lambda and must have Cloudwatch Logs permissions.

lambda_dynamodb_role

Description: An Amazon DynamoDB trigger that logs the updates made to a table.

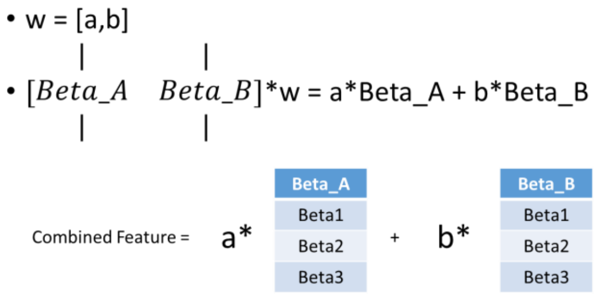
Your .py need to be same name as the handler. (from above figure, I need to set the name of my python code as dynamo.py)

```
import json
import boto3
import numpy as np
import decimal
from decimal import *
import time
print('Loading function')

def lambda_handler(event, context):
    tStart = time.time()
    ### get new sample size from dynamodb###
    dynamo = boto3.resource('dynamodb')
    table_S = dynamo.Table('SampleSize')
    db_response_S = table_S.scan()
    item_S = tuple(db_response_S['Items'])
    datanum = int(item_S[0]['Sample_size'])

    featurenum = 3
    collectornum = 2
    X = np.zeros((datanum,featurenum))
    y = np.zeros((datanum,1))
    betam = np.zeros((featurenum,collectornum))

    table_A = dynamo.Table('sensingdata_A')
```

```
table_B = dynamo.Table('sensingdata_B')
table_C = dynamo.Table('sensingdata_C')

db_response_A = table_A.scan()
item_A = tuple(db_response_A['Items'])


betam[0][0] = item_A[0]['feature_A']
betam[1][0] = item_A[0]['feature_B']
betam[2][0] = item_A[0]['feature_C']


db_response_B = table_B.scan()
item_B = tuple(db_response_B['Items'])


betam[0][1] = item_B[0]['feature_A']
betam[1][1] = item_B[0]['feature_B']
betam[2][1] = item_B[0]['feature_C']

db_response_C = table_C.scan()

item_C = tuple(db_response_C['Items'])
for i in range(datanum):
        X[i][0] = item_C[i]['X_1']
        X[i][1] = item_C[i]['X_2']
        X[i][2] = item_C[i]['X_3']
        y[i][0] = item_C[i]['Y']

data_bytes = item_C[1000]['data_bytes']
### store some result to dynamodb ###
table_LC = dynamo.Table('latency_C')
db_response_LC = table_LC.scan()
item_LC = tuple(db_response_LC['Items'])
Comm_pi_pi_A = item_A[0]['Comm_pi_pi']
Comm_pi_pi_B = item_B[0]['Comm_pi_pi']
Comm_pi_pi_C = item_LC[0]['Comm_pi_pi']
Comm_pi_lambda_A = item_A[0]['Comm_pi_lambda']
```

```
    Comm_pi_lambda_B = item_B[0]['Comm_pi_lambda']
    Comm_pi_lambda_C = item_LC[0]['Comm_pi_lambda']
    Compu_pi_A = item_A[0]['Compu_pi']
    Compu_pi_B = item_B[0]['Compu_pi']
    Compu_pi_C = item_LC[0]['ExecTime']
    Comm_pi_pi = np.max([Comm_pi_pi_A,Comm_pi_pi_B,Comm_pi_pi_C])
    Comm_pi_lambda =
np.max([Comm_pi_lambda_A,Comm_pi_lambda_B,Comm_pi_lambda_C])
    Compu_pi = np.max([Compu_pi_A,Compu_pi_B,Compu_pi_C])

    ### combine feature algorithm ###
    def prox_simplex(y):
        # projection onto simplex
        n = len(y)
        val = -np.sort(-y)
        suppt_v = np.cumsum(val) - np.arange(1, n+1, 1) * val
        k_act = np.sum(suppt_v < 1)
        lam = (np.sum(val[0:k_act]) - 1.0) / k_act
        x = np.maximum(y-lam, 0.0)
        return x

    def combine(y, X, betam):
        K = betam.shape[1]
        w = np.ones((K,)) / K
        maxit = 1000
        tol = 1e-3
        Xb = np.dot(X, betam)
        #print Xb
        step = 1.0 / np.max(np.linalg.svd(Xb, full_matrices=0, compute_uv=0)) ** 2

        for it in range(maxit):
            prev_w = np.copy(w)
            #print w
            res = y - np.dot(np.matrix(Xb),np.matrix(w).T)

            grad = -np.dot(np.matrix(Xb).T,np.matrix(res))
            w -= step * np.squeeze(np.asarray(grad.T))
            w = prox_simplex(w)
            #print(w)
            if np.linalg.norm(w - prev_w) / (1e-20 + np.linalg.norm(prev_w)) < tol:
                break

        return w
```

###The following figure shows you the concept of combining features:###



```
    w = combine(y,X,betam)
    w_temp = [decimal.Decimal(str(w[i]))for i in range(collectornum)]

    wb = np.dot(np.matrix(betam),np.matrix(w).T)
    Predict_y = np.dot(np.matrix(X),wb)
    Predict_y_array = np.squeeze(np.asarray(Predict_y))
    Predict_y_temp = decimal.Decimal(str(Predict_y_array[199]))
    ### need to change the type of the data to successfully store the data in to dynamodb ###
    MSE = np.sqrt(np.sum((y-np.squeeze(np.asarray(Predict_y)))**2))/datanum
    MSE_temp = decimal.Decimal(str(MSE))
    tEnd = time.time()
    Lambda_ExecTime = tEnd - tStart
    tEnd_temp = decimal.Decimal(str(tEnd))
    Lambda_ExecTime_temp = decimal.Decimal(str(Lambda_ExecTime))
    table = dynamo.Table('weightresult')
    #print Predict_y

    item = table.put_item(
        Item = {
            'environment' : 'roomA',
            'sensor': 'sensorA&B&C',
            'w_1' : w_temp[0],
            'w_2' : w_temp[1],
            'Prediction' : Predict_y_temp,
            #'Real_Result' : y[199],
            'Error' : MSE_temp,
            'Lambda_ExecTime' : Lambda_ExecTime_temp,
            'Time': tEnd_temp,
            'Comm_pi_pi': Comm_pi_pi,
            'Comm_pi_lambda': Comm_pi_lambda,
            'Compu_pi': Compu_pi,
            }
    )
```

Plot result:

```
from time import sleep
import csv
import json
import cPickle
import numpy as np
import paho.mqtt.client as paho
import os
import socket
import ssl
import json
import boto
from ConfigParser import NoOptionError
import time
import boto.dynamodb
import matplotlib.pylab as plt
import math



###connect to AWS dynamoDB###
connflag = False

conn = boto.dynamodb.connect_to_region(
    'us-east-2',
    aws_access_key_id='AKIAIALH5BDUW5WVA3FA',
    aws_secret_access_key='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')
```

```
def on_connect(client, userdata, flags, rc):
    global connflag
    connflag = True
    if rc==0:
        print ("Connection status: successful")
    elif rc==1:
        print ("Connection status: Connection refused")

##Bytes of data##
def utf8len(s):
    return len(s.encode('utf-8'))

#mqttc = paho.Client()
#mqttc.on_connect = on_connect
#
#awshost = "act744dbdaqsy.iot.us-east-2.amazonaws.com"
#awsport = 8883
##clientId = "MyRaspberryPi"
##thingName = "MyRaspberryPi"
#caPath = "D:\Princeton testbed\\Final\\deviceSDK\\root-CA.pem.crt"
#certPath = "D:\Princeton testbed\\Final\\deviceSDK\\9ad1cc85e6-certificate.pem.crt"
#keyPath = "D:\Princeton testbed\\Final\\deviceSDK\\9ad1cc85e6-private.pem.key"
#
#mqttc.tls_set(caPath, certfile=certPath, keyfile=keyPath, cert_reqs=ssl.CERT_REQUIRED, tls_version=ssl.PROTOCOL_TLSv1_2, ciphers=None)
#mqttc.connect(awshost, awsport, keepalive=60)
#
#mqttc.loop_start()

Time_old = 0
###see if the tables are updated
while True:
    table_1 = conn.get_table('latency_C')
    table_2 = conn.get_table('weightresult')
    table_4 = conn.get_table('sensingdata_C')
    while True:
        item_2 = table_2.get_item(
            # Our hash key is 'forum'
            hash_key='roomA',
            # Our range key is 'subject'
            range_key='sensorA&B&C',
            # This has the
            #attrs=item_data
            )
        if Time_old != item_2['Time']:
            Time_old = item_2['Time']
            break
        else:
            Time_old = item_2['Time']
```

<u>*The next step is getting items from table, just the same as Sensor Pi.</u>**

```
fig, axs =plt.subplots(3,1) # you can change the number of subplot in a figure window
fig.set_size_inches(20,2) # you can change the table size if you want

###plot table###
latency_collabel= ["Data_Amount","Communication_Latency","Communication_Latency","Computation_Latency","Computation_Latency","Total_Latency"]

latency_data = [["(Bytes)","Sensor_Pi->Gateway_Pi","Gateway_Pi->Lambda","Gateway_Pi","Lambda","(second)"],[data_bytes,Comm_pi_pi,Comm_pi_lambda,Compu_pi,Compu_lambda,Total_Latency],[" "," "," "," "," "," "]]

colors = ['gray','#DDDDFF','#DDDDFF','#FFD0DF','#FFD0DF','#FDFFD0']
axs[0].axis('tight')
axs[0].axis('off') #without axis
the_table_latency = axs[0].table(cellText=latency_data,colLabels=latency_collabel,loc='center',colColours=colors)
the_table_latency.scale(1.3,1.5)
```