This document explains how to get iot data in real time from the cloud in the program

realtimegraph - alllines.py and then make an animated graph which displays this real time data. This also applies to the realtimedata – oneline.py which was used as a trial to graph a single line and then extended to this python code. The realtimedata – movingxaxis is also a trial to move the x-axis for a single line which was then extended to this code and applied for all lines.

The program realtimegraph – oneline.py does the same but only plots a single line. This was done as a trial version.

* The resource that we want to work with is specified as a resource.
* The table name IMU\_Data is specified with the Table parameter to write data into it.
* The table.scan command scans the entire table and gets the data of all the attributes.

Documentation for scan : <https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/dynamodb.html#DynamoDB.Table.scan>

Api Reference:

<https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_Scan.html>

Stack Overflow:

<https://stackoverflow.com/questions/36780856/complete-scan-of-dynamodb-with-boto3>

* The set of variables that will hold the data to be plotted are initialised with an empty list, so that values can be appended to them.
* fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, figsize=(10, 15))

This creates 3 plots one below the other in a single figure to plot the data and that the x-axis is common.

* The scan command returns a dictionary with a lot of values(mentioned in docs above) which is stored in response.
* In the try block, we get the last set of data that we just retrieved. If there is no data, then there will be an index error when we try to do this which we catch with the except and exit the program.
* From the last set of data, we get the primary keys of the table which are ‘device\_id’, and ‘timestamp’ so that we can pass it in the next iteration.
* The “Count” parameter is checked to see how many data points were received. If it is more than 20, then displaying it in the graph will be very congested and will also slow the system down. So, if it is more than 20, then only the last 25 data points are taken to be displayed.
* Then, the sleep makes the program wait for x seconds so that the next set of data is entered into the db, because data is being sent at x time interval, so we have to wait x secs before the next read, otherwise we’ll get empty data and the program will terminate.
* Now, the FuncAnimation function is called. For the tutorial followed to create the animations, refer: <https://youtu.be/ZmYPzESC5YY>
* To this function, the figure ‘fig’ that we created earlier is passed along with the name of the function that will plot the data ‘animate’. The interval mentioned here, means that the ‘animate’ function will be called again and again in that interval.
* Now, the animate(i) (not clear why i is received) function is called. The response and lastevalkey are declared as global as their values outside of this function must be used here for the initial data read.
* Now , we get the initial data, i.e, the data that we read before this function was called. The data from the table is under the key ‘Items’ inside the dictionary in the response variable.
* The initial read is done outside the function because, if we place the same code in the animate function, then we have to mention an ExclusiveStartKey for the first read itself which is not possible( work out the flow if it seems confusing).
* Here, timestamp alone uses the extend function because new timestamps from the new data will be appended to the existing list of timestamps from the old data.
* The timestamp is split and only the time part is taken using the split() which will return a list with the date at index 0 and time at index 1. So, we specify [1] to get the time. Then in the time, we don’t want to display the milliseconds as it will be too much clutter. So, [:-4], will make us get until the “end-4” index of the time, which ignores the milliseconds part.
* Now, each data list is sent to split\_axes function to split into x,y, and z axis data to make it easier to plot along with the individual axis variable into which it should be stored.
* All the coordinate data is in the form a string in this form [‘[0, 0, 0]’,’[1, 2, 3]’, …..]. So, we take each set from the list and then strip the brackets( ][ ). The, there is a space before the numbers, so we remove that with replace and then split them by the commas and store them in a list. So, [‘[0, 0, 0]’,’[1, 2, 3]’, …..]. will become [[‘0’,’0’,’0’], [‘1’,’2’,’3’],……]
* Now the string numbers should be converted to float in the next list comprehension.
* All the numbers in the 0th position(i.e, the x-axis) are stored in x and so on.
* These are stored in mag\_x, mag\_y etc, according to the variables which were passed.
* The extend() function joins two lists together. This is used because when the next set of data is received, that data should be appended to the already existing data.
* Then the data is plotted.
* After which we check whether there are more than 20 data points to be displayed. If yes, then the move\_axis() function is called in order to remove the older data to make space in the graph for newer data. If this is not done, the graph eventually is congested with a lot of data and it also makes the program slow to plot the new data.
* In move\_axis(), the first two data points inside the x (timestamp) and y(sensor) variables are removed. This way, when new data comes in, the old data is consistently removed.
* Coming back to animate(), we read the next set of data again but this time ExclusiveStartKey = lastevalkey is specified. What this does is, it reads data starting tight after that particular key. So, we don’t have read the old data again and only read the new data that has entered the table.
* If no new data has entered the table, then in the try block, we’ll get an IndexError when we try to get the last item. So, we catch that exception and break out of the loop.
* Otherwise, the lastevalkey is recalculated and the program proceeds.
* Next, we clear all the 3 plots because if we don’t , then everytime this function is called by FuncAnimation, the lines will be redrawn over each other causing huge computational load.
* Then, the graphs are plotted.
* The next time, this animate is called, all the x and y axes data is update, so we clear the current plot and redraw the lines with the new data which gives us a graph that changes dynamically with incoming data.
* The ply.show() is what displays the window in which we can see the graph.