Internet of Things: Architecture, Application and Implementation
CSC/ECE 591/792

Arpita Awasthi, Haris Ansari, Mathioli Ramalingam, Ravikanth Somsole

Design Choices

Following are the design choices we had to make at different stages of the project:

1) Selection of data from the gyro output:
We used the gyro-Y output from the sensor. We stick the sensor with its Y axis parallel to the door and perpendicular to the ground and Z axis perpendicular to the door. We considered a threshold value on gyro-Y axis, which is 1000 for opening event and -1000 for closing event. Creating a libSVM model out of this data would thus give the best prediction results.

2) Obtaining Training Data:
We trained the data using a small door, with the sensor connected 10 inches away from the hinge. We took samples while opening the door, closing the door, and while it is stationary. We filtered the some values from the captured data, specifically the values at the edge of a closing/opening operation, which are "noise" points, and thus do not represent the opening/closing operation of the door. We took around 50 samples each for when the door is in opening phase, closing phase, and when it is stationary. After analyzing the data, we kept the threshold on gyro_Y value to distinguish between opening and closing event. As we have six attributes (gyro_X, gyro_Y, gyro_Z, acc_X, acc_Y, acc_Z), we averaged out every attribute values and condensed to two rows inside our training data. First row corresponds to class label 1 (Opening event) and second row corresponds to class label -1 (Closing event). We even tested our samples on Weka where it was showing clear distinction between the two classes.

3) Training and creating libSVM model:
We used grid.py tool, which is a parameter selection tool for SVM classification using RBF kernel, provided by libSVM. We used the default 5 fold cross validation scheme, since we did not have a very complicated set of data. We thus obtained the c and gamma values for the SVM model we created from our training samples.
The command we executed was : python grid.py training_data
For the training data, we thus arrived at the values : c = 2.0 and gamma = 3.051e-05

4) Real-time analytics
We pushed the libsvm files to the Bluemix and created the executables that we used to analyze the real-time sensor values in our Server code inside Bluemix. As we already have the trained data, we pushed it to Bluemix. Now, every time when the door is moving or kept stationary, the sensor values are continuously published to Bluemix where the model is created on-the-fly and these real-time values are tested on this model giving out the result inside “result” file.

5) Events and MQTT Clients

We created three events or topics which our Raspberry Pi publishes to the Bluemix Server Application named: Opening, Closing, and Still. "***Opening***" event publishes the attribute values of gyro_X, gyro_Y, gyro_Z, acc_X, acc_Y, acc_Z considering threshold of 1000 on gyro_Y axis. On the other hand, "***Closing***" event publishes the attribute values of gyro_X, gyro_Y, gyro_Z, acc_X, acc_Y, acc_Z considering threshold of -1000 on gyro_Y axis. Rest all are the values when the door is stationary which is considered as "***Still***" event. Hence, Raspberry Pi is the publisher of these three events. Bluemix Server Application is the subscriber to these events and publishes "***Status***" event which gives the decision (either door open or close). Laptop 2 is the subscriber to the event "***Status***".

We used ibmiotf library in Python to develop MQTT protocol between the clients.

6) MQTT Broker:

Since we use IBM Bluemix, there is a broker service already provided in it, and hence we do not need to set up

Schematic Diagram