AN12766

Anomaly Detection with eIQ using K-Means clustering in Tensor Flow Lite

Rev. 1 — 11/ 2020 Application Note

Contents

1 Introduction

This document provides step by step instruction to enable a machine condition monitoring application using anomaly detection. The application creates the model using k-means clustering algorithm with TensorFlow framework and Python. Once the model is trained and developed, then it is converted to TensorFlow Lite and finally to header file so that it can be exported to i.MX RT1050 board.

2 Applications support

NXP eIQ Software Development Environment (SDE) can support several types of applications such as:

- Facial recognition
- Anomaly detection
- Object detection
- Voice recognition
- Image processing
- Sensor drivers
- Vision apps
- Audio front end

This application note focuses on anomaly detection by collecting accelerometer data for machine condition monitoring.

3 Application setup

The i.MX RT1050 EVKB is mounted on a table fan to collect the vibration data as shown in Figure 1 for the training data. The data is exported to an XLS sheet for different conditions. The collected data is the raw X and Y data on which we require to do further analysis. We must get the real-time data of the machine for monitoring its condition in healthy and faulty state. Use the FXOS8700 accelerometer available on the i.MXRT1050 which communicates with i.MX RT1050 over I2C to monitor the machine health.

3.1 Hardware

NXP i.MX RT1050 EVKB

Figure 1. i.MX RT board setup

3.2 Software

Use the SDK_2.6.0_EVKB-IMXRT1050 or later.

The application monitors the fan to detect anomalies in vibration. For the training data, the i.MXRT1050 EVKB is mounted on a table fan to collect the vibration data.

The data is exported to an XLS sheet from the serial console for different conditions by a Python script. The collected data is the raw X, Y, and Z axes data on which we require to do further analysis.

4 Model selection

Once the application and requirement is in place, select the type of learnings for the model that best suits the application. The different types of learnings are as follows:

- Supervised learning: Training data includes desired outputs
- Unsupervised learning: Training data does not include desired outputs
- Semi-supervised learning: Training data includes a few desired outputs
- Reinforcement learning: Rewards from a sequence of actions

Referred the several white papers for anomaly detection and found that most of them are using unsupervised or semi supervised learning for this kind of application. So decided to try both and then conclude which suits best for the application.

The application of anomaly detection, first selects unsupervised learning to train the model with only healthy data. If any data is received out of healthy condition, it detectes as anomaly. But the drawback of the approach is that the model detects anomaly for very less change in vibrations and it is difficult to create such an atmosphere where there is no change in vibrations.

Due to the above reason, switch to the semi-supervised learning which includes some anomalies along with the healthy data for training the model. So in this case the model is trained with both healthy and anomalous data and it detects anomalies of all types.

References

- https://www.intechopen.com/books/artificial-intelligence-emerging-trends-and-applications/artificial-intelligence-application-in-machine-condition-monitoring-and-fault-diagnosis
- http://wiredspace.wits.ac.za/bitstream/handle/10539/5482/VILAKAZI_DISSERTATION.pdf?sequence=1&isAllowed=y
- https://towardsdatascience.com/machine-learning-for-anomaly-detection-and-condition-monitoring-d4614e7de770

5 Data collection

The vibration data which is used to train the model must be analyzed before applying it to the model. The first step is to collect the real-time vibration data. As semi-supervised learning is selected to use the application (Data_collection.py script) which is developed for data collection and to create anomalies in between by tapping from different sides.

The application for data collection integrates the FXOS800CQ configured with the following settings:

- Mode: Accelerometer only
- Output Data Rate: 400 Hz
- I2C Baud Rate: 1 MBPS
- I2C Read Rate: 1.5 Ms

The application constantly reads the vibration data from accelerometer over I2C as per the read rate and dump the data on serial console for all the three X, Y, and Z axes. Python script is developed “Data_collection.py” to read the data from serial console, parses it and creates a workbook where it copies this data of all three axes in separate columns. Configure the amount of time which is required to collect the data. The script collects the data for that much time and then automatically closes the console and save the data. The process can be killed at any time as the sheet is saved at every 1 minute, so that it does not lose data in case the program is killed before the configured time.

6 Data analysis

During collection of this data, keep a track of the anomalies; for example, the number of times anomalies are created in one sheet, total number of anomalies created, direction of anomaly. Once the data is collected, it must be analyzed to decide to filtering and feature extraction method required before feeding it to the model. And do the frequency and time domain analysis of the collected data.

6.1 Time domain analysis

In the time domain analysis, plot the graph of captured data in the sheet and observe the peaks of data. And find that RMS value of waveform by doing time domain analysis of vibration data. It is a useful feature to detect anomaly in the vibration.

6.2 Frequency domain analysis

For the frequency domain analysis, a Python script is written to generate the Fast Fourier Transform (FFT) of original and filtered data. Find the script for FFT in the release package under “Scripts” folder. By Observing the harmonics of healthy and anomalous data from the FFT graphs, the overall vibration frequency ranges from 100 – 300 Hz. As per the Nyquist criteria, the sample rate must be greater than 2 times of the frequency. So as per this calculation keeps the sample rate as 666.66 Hz which leads to 1.5

milliseconds of data collection. And also observe few harmonics which must be removed for effective feature abstraction from vibration data.

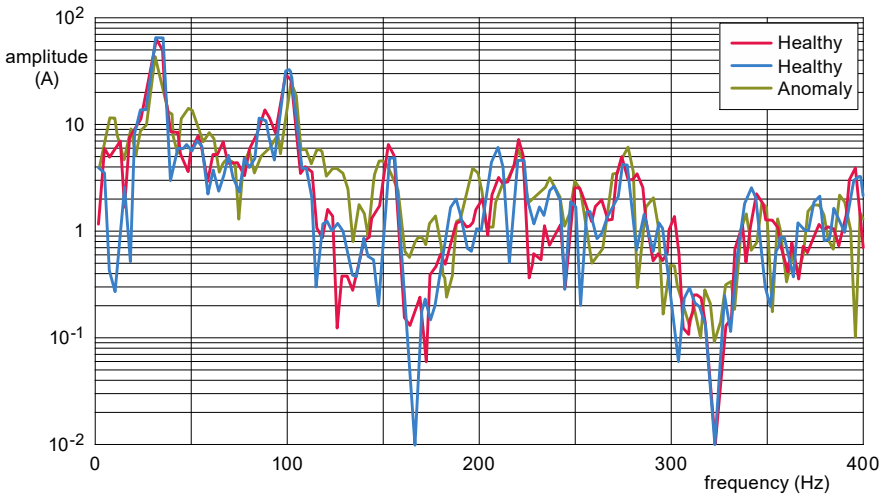


Figure 2. X-axis filtered

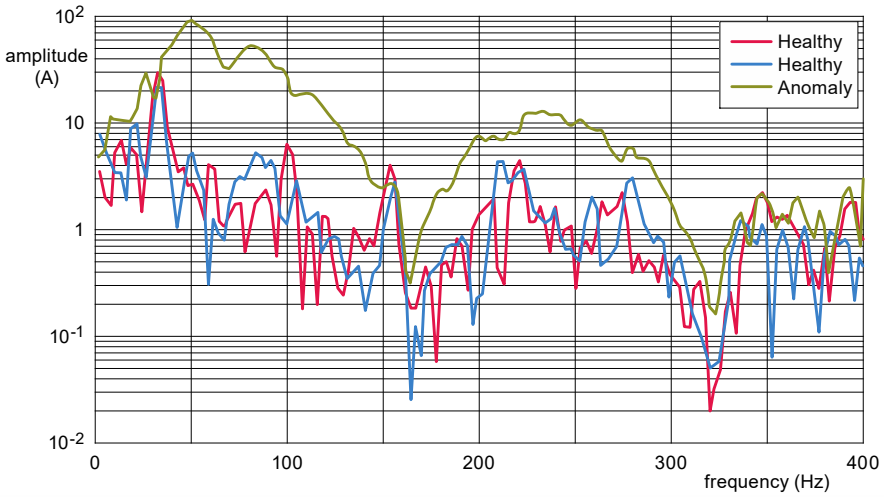


Figure 3. Y-axis filtered

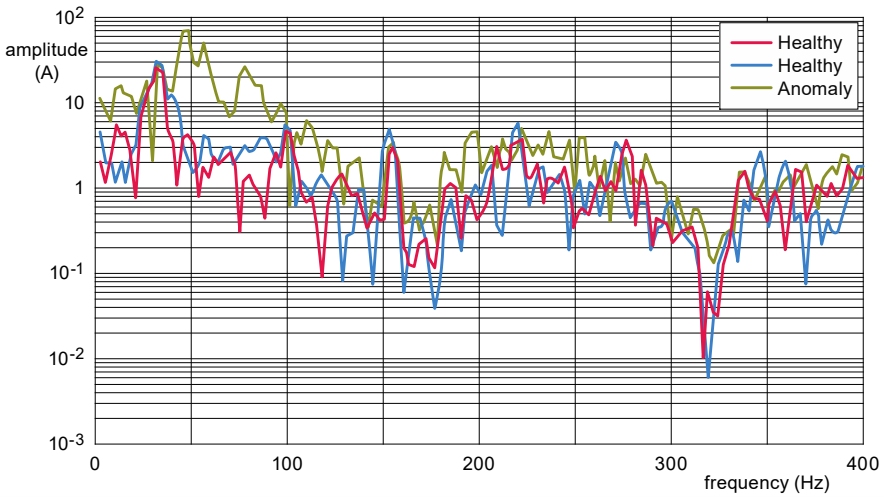


Figure 4. Z-axis filtered

7 Data pre-processing

7.1 Low-Pass filtering

Once the data is collected, it must be filtered which removes the unwanted noise and glitches from the data. After the frequency domain analysis, it is decided to apply the moving average filter which takes the average of 5 samples. And plot the graph of the filtered data, it is observed that the data is much smoother and all unwanted glitches are removed. This graph was plotted in the excel sheet shown in Figure 5.

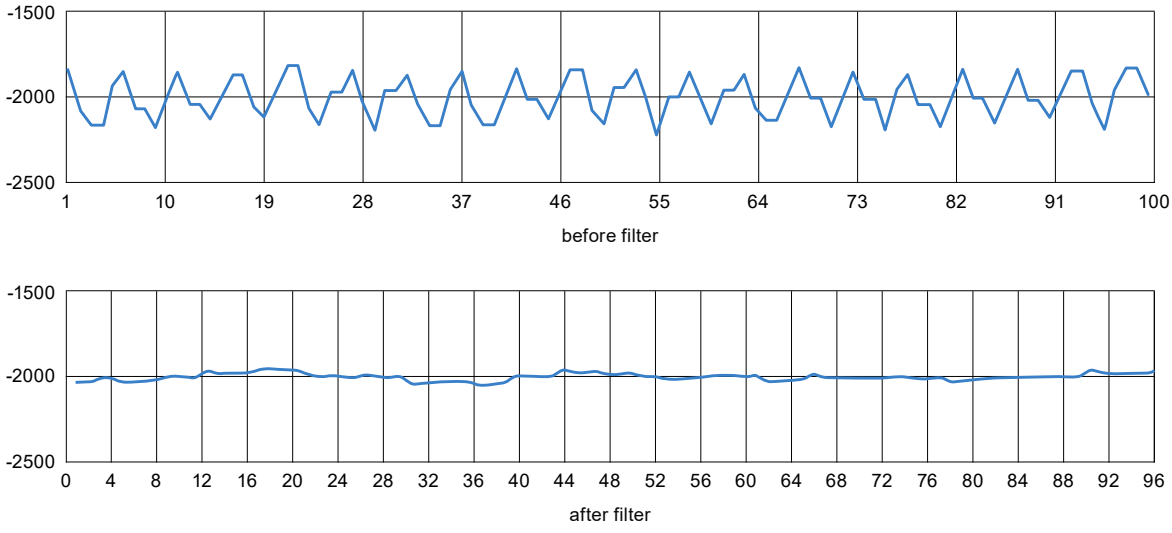


Figure 5. Low-pass filer graph

8 Feature extraction

After the filtering process, the smooth data is ready for feature extraction. By doing time domain analysis of vibration data, the RMS value of waveform is found to be a useful feature to detect the anomaly in the vibration. The RMS is of 10 samples. And this number

is decided on trial and error basis after observing the results for different values. The number is to be decided in such a way that the difference between RMS values of healthy data and anomalous data is large enough to be easily separated by the model.

The application reads the accelerometer raw data of X, Y, and Z axes at every 1.5 ms and sends it to PC over UART interface.

Next a Python script is developed to read the data from UART and parses it to get the data of X, Y, and Z axes and dump it in xls file.

After successfully creating the XLS file with parsed X, Y, and Z axes data, prepare an insertion chart for all the three axes to analyze the data. And run the test many times and collect the data in different conditions like normal condition and anomalous condition like taping in between, moving the board horizontally, moving the board vertically. After collecting the data in different conditions, analyze it manually by plotting the graphs and capturing the anomalies.

Model development with TensorFlow is shown in Figure 6.

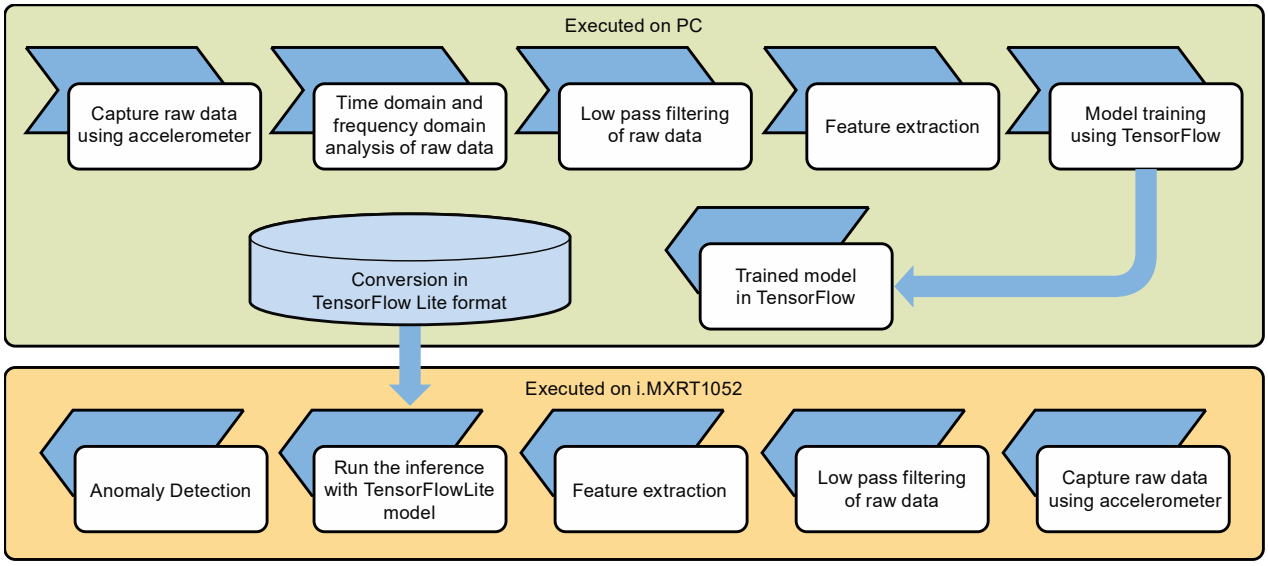


Figure 6. Model development with Tensorflow

9 Model development in TensorFlow

In this demo, K-Means Clustering algorithm is used for anomaly detection. The model is developed in Python using TensorFlow. It takes raw accelerometer data as input and extracts different features from it by calculating RMS. After analyzing the Data, it is fed to the K-Means clustering which provides output in terms of clusters.

Train the model as per the requirement of the application in Python. The model should be developed in TensorFlow, so that it can be converted into TensorFlow Lite format for i.MXRT1050 processing.

The model in Python should be developed keeping in mind the input to be fed and the output expected. And, same model is saved as .pb format, converted to .tflite format and imported on the application running on i.MXRT1050.

When the model run on i.MXRT1050, it is expected to feed the input in same format and size as developed in the Python. For example, look at the developed model for anomaly detection with this package, the input size is 2005. This is an empirical number which is fixed on several trials and errors while developing the model in Python. Based on several trials the conclusion is that the model gives the expected output if the input size is 200 samples. Before feeding the samples to model, apply low pass filter to the data which is done using moving average. After that calculate RMS values of samples (10 samples). Low pass filter and RMS calculation are provided below.

1. Applying low pass filter to data:

   Moving average:

   $A_1 = (x1+x2+x3+x4+x5)/5$

A_2 = (x2+x3+x4+x5+x6)/5

goes till A_n

Apply low pass filter by taking moving average of 5 samples so,

Output size = (input size – 5)

2. Calculating RMS values of filtered data:

   Final output size = (size of input to RMS)/10

   Apply RMS on filtered data:

   Where, the output of filter = Input of RMS

   Hence:

   Final output size = (input size -5)/10

Now this final output after calculating RMS should be provided as input to the model. The model requires 200 inputs then only it gives expected output.

For the final output size to be 200, from the above calculations the input size of raw sample should be 2005. So, 2005 raw samples are taken as input in the application.

9.1 Training and saving the model

The developed model must be trained with healthy and some anomalies data and saved using the SavedModel in TensorFlow. First train the model and get the trained data. In the example, this is done using “kmeans_development_training.py” and as per the algorithm of kmeans clustering take the trained data as centroids.

This trained data must be saved to the model that is running on the i.MX RT1050 platform. This model must be given the data that is filtered and analyzed as the input and then saved in the SavedModel format. This is performed by the script *Export_Trained_model.py*.

The SavedModel saves the model in ProtocolBuffer (.pb) format. The APIs for SavedModel are as below:

1. tf.compat.v1.saved_model.simple_save
2. tf.saved_model.builder.SavedModelBuilder&builder.add_meta_graph_and_variables

For further details about the SavedModel format, go through the link below:

https://www.tensorflow.org/guide/saved_model

9.2 Converting model in TensorFlow Lite format

The trained and saved model must be converted into a TensorFlow Lite compatible format. Follow the steps below to do so:

1. Convert the saved model to tflite format:

   export_dir = "<Path to saved model>/savedmodel"

   converter = tf.lite.TFLiteConverter.from_saved_model(export_dir)

   tflite_model = converter.convert()

   open("converted_model.tflite", "wb").write(tflite_model)

2. Convert the tflite file to converted_model.h file

   It can be done using the xxd utility like below:

   xxd -i ./converted_model.tflite> ./converted_model.h

   The model is ready in TensorFlow Lite format. Now add “converted_model.h” header file in the Tensorflow Lite application project. Now, project must have below structure as shown in Figure 7.

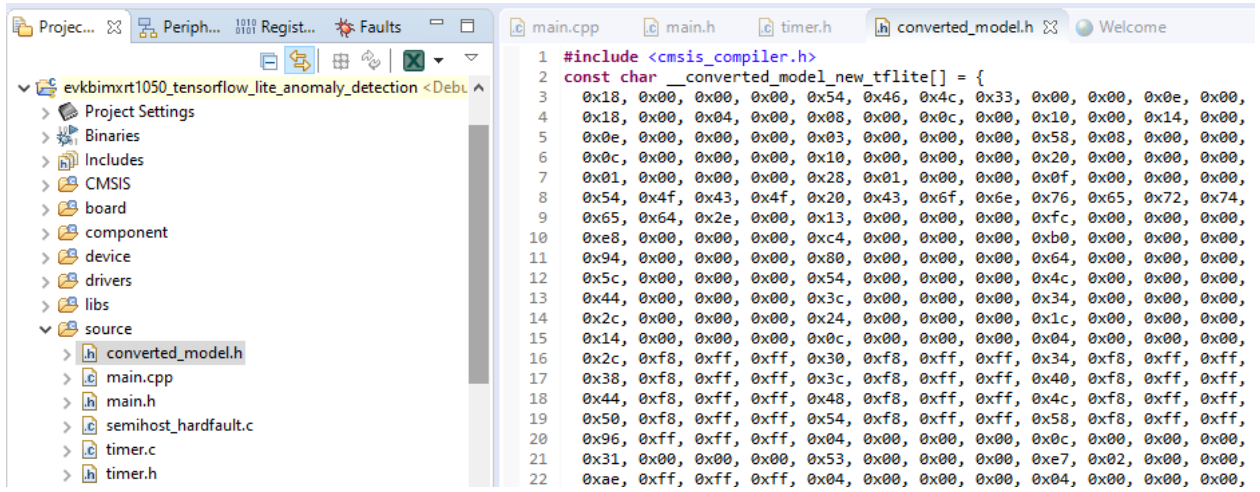


Figure 7. Anomaly detection model file in application project

9.3 Model verification using TensorFlow

At this point, the model is trained in .pb (Protocol Buffer) format. Now convert this trained model to TensorFlow Lite format. It can be done using the script “conversion.py” in the package. This script provides the output in the form of “.tflite” file. This file is the converted model in TensorFlow Lite format. Before running this model on edge using TensorFlow Lite inference, it is necessary to verify the model on the machine using TensorFlow.

Find the steps below for running a “.tflite” file using TensorFlow:

1. Loading the TensorFlow Lite model:

   tf.lite.Interpreter
2. Get Input & Output TensorFlow from the model:

   interpreter.get_input_details()

   interpreter.get_output_details()
3. Provide the new input
4. Reshape the input in required shape and type
5. Run the loaded model with new input:

   interpreter.invoke()

Validate the model by providing different inputs and observe the outputs. And, refer to the script “check_tflite.py” which is an example for the current model of anomaly detection. When check_tflite.py is run, it produces output (distance of each point from each centroids). Value for each point are mentioned in check_tflite.py and centroid value is fetch from trained model file. By mathematical calculation, find the distance between each point and centroids.

In this script, the above mentioned steps are followed. Find the script at the location below:

<path-to-ML-release-Package>/scripts/check_tflite.py

10 Application development on TensorFlow Lite using eIQ

For developing any application on TensorFlow Lite using eIQ SDE, take the reference of the available demo application. And, find the demo applications from the i.MX RT1050 SDK which can be downloaded from https://mcuxpresso.nxp.com.

For basic steps of developing any application on microcontroller using TensorFlow Lite, find the link below:

https://www.tensorflow.org/lite/microcontrollers/overview.

11 Important notes regarding eIQ SDE

There are some operators on TensorFlow which supports higher versions of TensorFlow Lite but do not support the TensorFlow Lite version (r1.11) that eIQ SDE supports. Find the points below:

- The tf.square operator is not a built-in operator in TensorFlow Lite version supported by the eIQ SDE and the model requires this operator.

Workaround:

As a workaround of the above unsupported operator, multiplication is used to perform square currently:

- The tf.argmin is a built-in operator of TensorFlow Lite. But found some unexpected behavior in the output that is included in tf.argmin as a part of TensorFlow model and run it on TensorFlow Lite after necessary conversions, the dimensions of the output changes. For example, if a tensor with dimension (3,409) as input is given to tf.argmin (tensor,axis=0) then it must provide an output tensor with shape 409. This is providing the expected output on TensorFlow but if it is converted to .tflite and then to .h file and run the model on TensorFlow Lite on i.MXRT1050 the output dimension is seen as (3,1) instead of (409,1).

Workaround:

The logic of tf.armin is developed on the application for anomaly detection which runs on i.MX RT1050 and able to detect the clusters with an anomaly on i.MX RT1050 board. The same behavior is observed for tf.argmax operator.

12 Conclusion

This exercise demonstrates, how model is trained by using TensorFlow framework by collecting the sensor data from i.MX RT1050 board on computer through Python script. And also demonstrate how trained model is converted to the supporting file which can be imported to i.MX RT1050 board.

This provides broader view to user, that how TensorFlow framework can be implemented on embedded board.

13 Revision history

Rev. Number	Date	Substantative Changes
0	04/2020	Initial release
1	11/2020	Added the appendix

A eIQ Anomaly Detection Custom Case Step by Step

A.1 General instructions

Anomaly detection gives steps for training the model and converting it to TensorFlow Lite and finally to header file which is exported to application project for i.MX RT1050 board.

This document also describes the minimum package requirements for running Python scripts.

A.1.1 Prerequisites

There are several packages that must be installed, as per requirement by the Python scripts. Python must be installed in the system. Other packages which are required by Python are mentioned below. Python packages installation command may vary, depending upon Python version installed in the system.

1. Update the Python installer tool which is called pip. All the commands are run on the Windows command prompt.

```
python -m pip install -U pip
```

2. Install the TensorFlow libraries and support for Python.

```
python -m pip install TensorFlow
```

3. Install other useful Python packages. Not all of these are used for the lab, but are useful for other eIQ demos and scripts.

```
python -m pip install tensorflow-datasets
python -m pip install xlwt python-tk pandas
python -m pip install numpy scipy matplotlib ipython jupyter     sympy nose
python -m pip install opencv-python
python -m pip install PILLOW
python -m pip install pyserial
```

4. If the operating system is Windows, add the following directories to executable PATH in Windows environment, if not already installed:

```
<python install directory>/scripts
```

5. Install vim v8.1 or above for conversion of .tflite file into .h file. If on Windows, install Vim 8.1 from vim.org

   There is a binary convertor program named xxd.exe located inside that package that is needed.

6. If the operating system is Windows, add the following directories to executable PATH in windows environment, if not already installed.

```
<vim_install_directory>
```

7. Verify that the PATH is set correctly by typing “tflite_convert” and “xxd -v” in a command prompt. There must not be any errors about an unrecognized command.

A.1.2 Steps for training the model and converting it to TensorFlow Lite format

A.1.2.1 Data collection

1. Setup the EVKB i.MX RT1050 on fan to capture its vibrations on the position desired for the trained model. Make sure that the accelerometer (FXOS800CQ) is in place at U32 as shown in Figure 8:

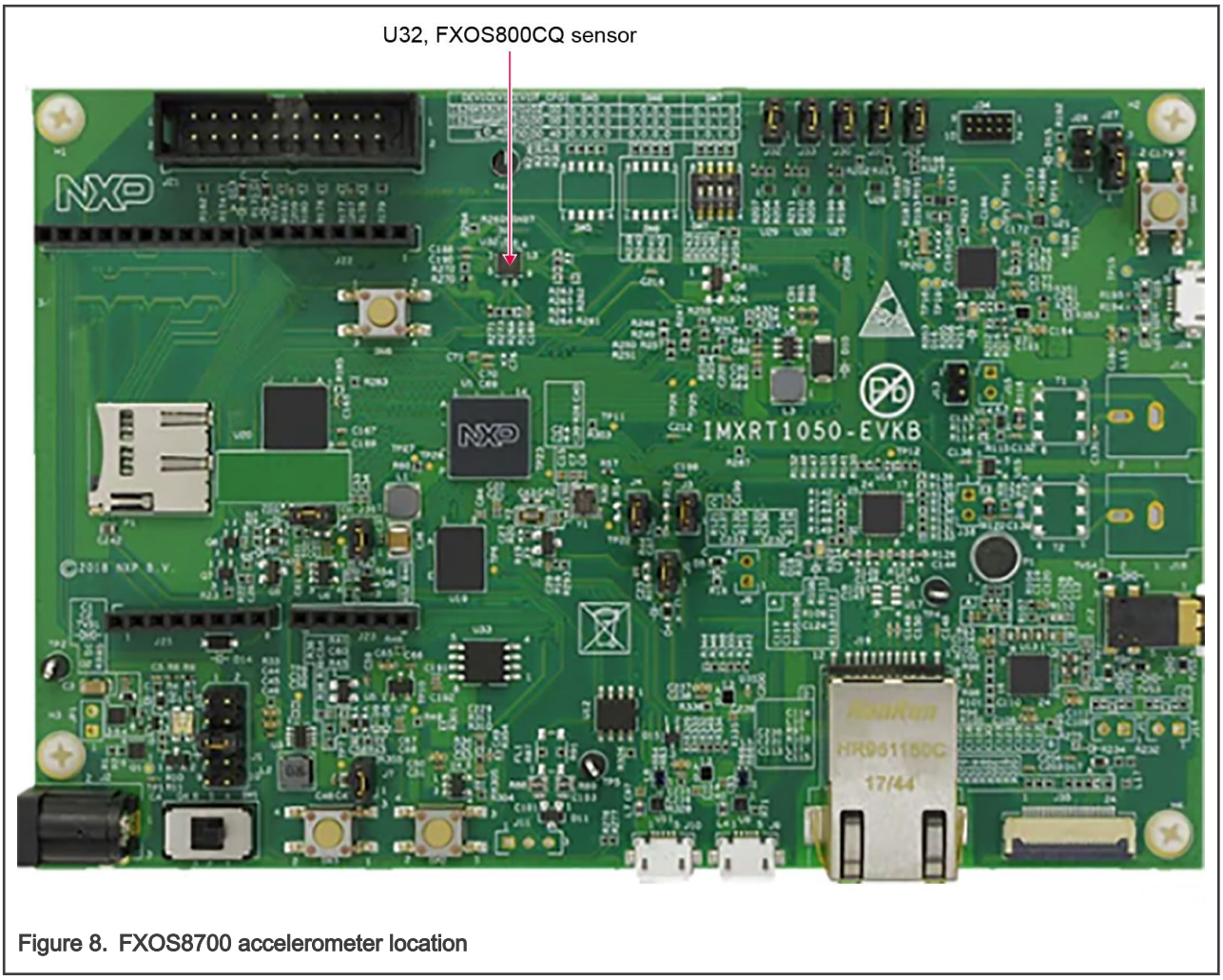


Figure 8. FXOS8700 accelerometer location

2. Flash the application ”evkbimxrt1050_tensorflow_lite_anomaly_detection” with macros disabled **#define RUN_INFERENCE** in timer.h file for data collection and must enable for running inference in application. Make sure that the serial console is configured with parameters as shown in Figure 9.

   For the port connection, check your device manager and see which COM port has been assigned. Start getting X, Y, and Z axes data on serial console.

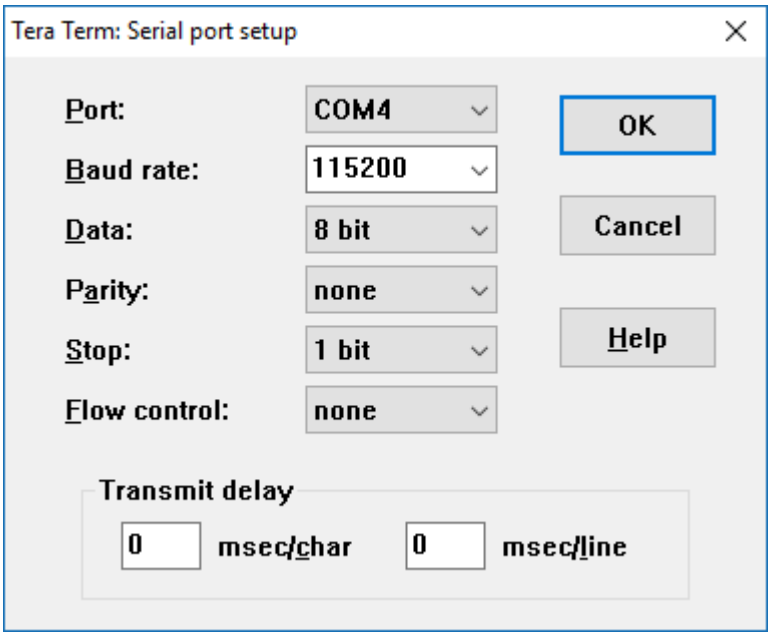


Figure 9. Tera Term serial console configuration

3. Close the console and run the script “Data_Collection.py” from MLPC folder. The script can be modified for the amount of time used to capture the data as follows:

   a. Hours_ = 1

```
##########################################################################
# Timming Configurations
##########################################################################
Hours_ = 1
```

Figure 10. Timing configuration

   b. After completion of the configured time period, the script ends and a workbook appears in the folder scripts/DataBase with the name of “Data_Collection_<time>.xls”.

NOTE

While Data_Collection.py script is running, time to time shake or tap i.MX RT board, so that bad data (anomalous data) also collected along with healthy data

4. Once Data collection workbook is available in folder scripts/Database, run the “kmeans_development_training.py” from MLPC folder. The complete execution of kmeans_development_training.py script consumes decent time depending upon system configuration.

A.1.2.2 Training the model

Once the training is performed (executing the kmeans_development_training.py), there must be 9 trained values, 3 values for each X, Y, and Z axes as shown in Figure 11.

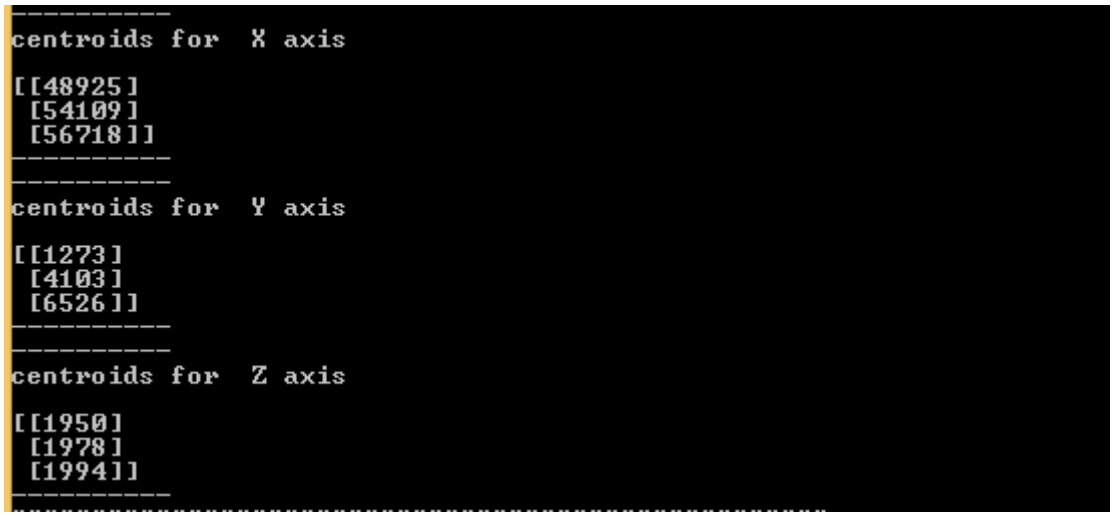


Figure 11. Centroids value for X, Y, and Z axis

A.1.2.3 Export the trained model

1. Feed the 9 values (as shown in Figure - 4) to the script “Export_trained_data_model.py” as follows:

```
#x-axis
input_datax = np.array([48925,54109,56718])
#y-axis
input_datay = np.array([1273,4103,6526])
#z- axis
input_dataz = np.array([1950,1978,1994])
```

2. Run the script “Export_trained_data_model.py” from MLPC folder. And find the “.pb” file in Saved_Model folder as exported trained model which must be converted to .tflite. The exported model is converted to .tflite using the script “conversion.py”.

A.1.2.4 Conversion to TensorFlow Lite

1. Run the script “conversion.py” from MLPC folder.
2. Get a file with .tflite extension in TFlite_Model folder which is the converted model to TensorFlow Lite format.
3. Open the Windows command prompt. Navigate to “..\ML_Release\Scripts\TFlite_Model” folder and execute the following command to convert the .tflite file to .h format.

```
xxd -i ./converted_model_new.tflite > ./converted_model.h
```

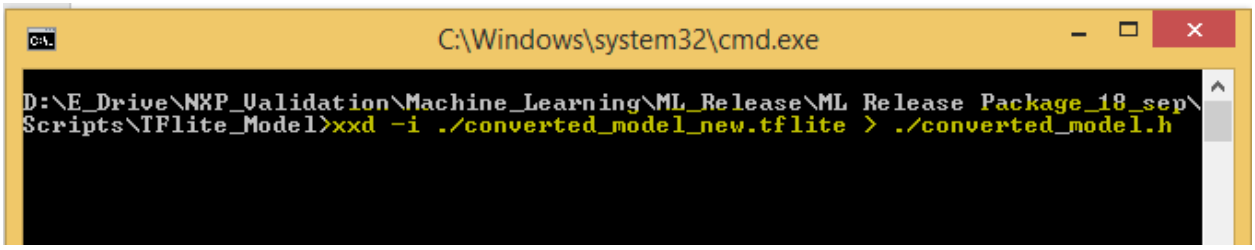


Figure 12. Instruction for model conversion to converted_model.h

4. After converted_model.h file is generated, change “unsigned char” to “const char” in converted_model.h file, as shown in the following figure:

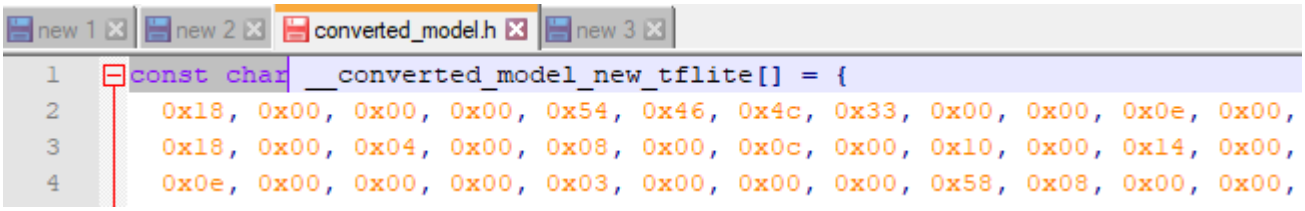


Figure 13. Edited converted model file

A.1.2.5 Adding the converted model header file to application project

1. Add converted_model.h header file in the TensorFlow Lite application project. The file must look like as shown in the Figure 14.

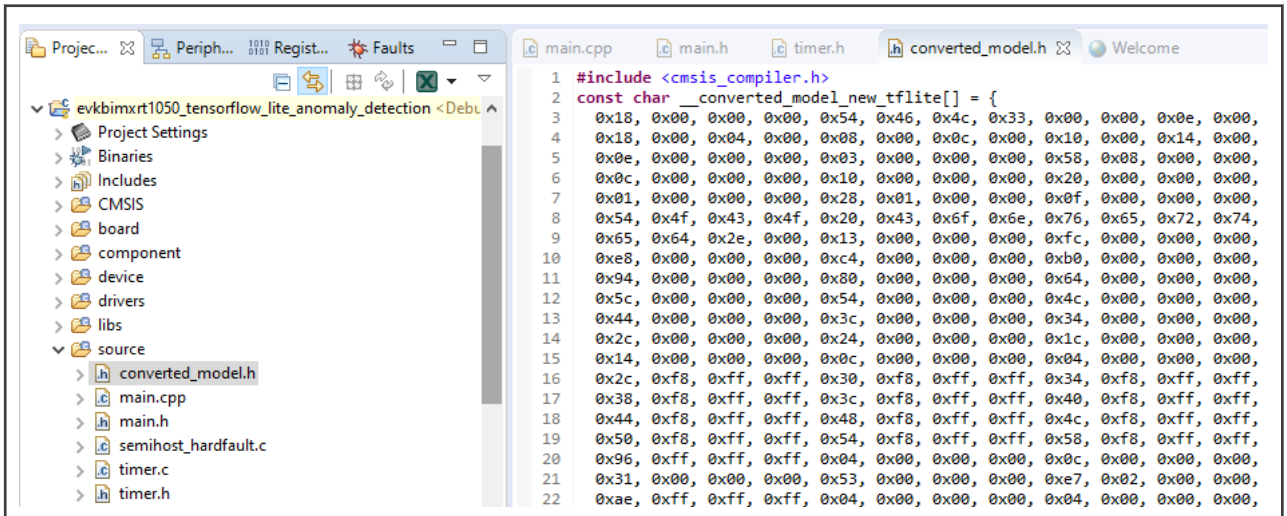


Figure 14. Anomaly detection model file in application project

2. Enable the **#define RUN_INFERENCE** in timer.h file, and replace the model buffer name in main.cpp file as shown below.

```
tflite::FlatBufferModel::BuildFromBuffer(model_data,model_data_len);
```

```
130
131   model = tflite::FlatBufferModel::BuildFromBuffer(__converted_model_new_tflite, __converted_model_new_tflite_len);
132
```

Figure 15. Replace model data buffer name in main.cpp

3. After flashing the binary on i.MX RT1050 board, view the output on serial console as shown in the Figure 16. View output on GUI by running Display.py script.

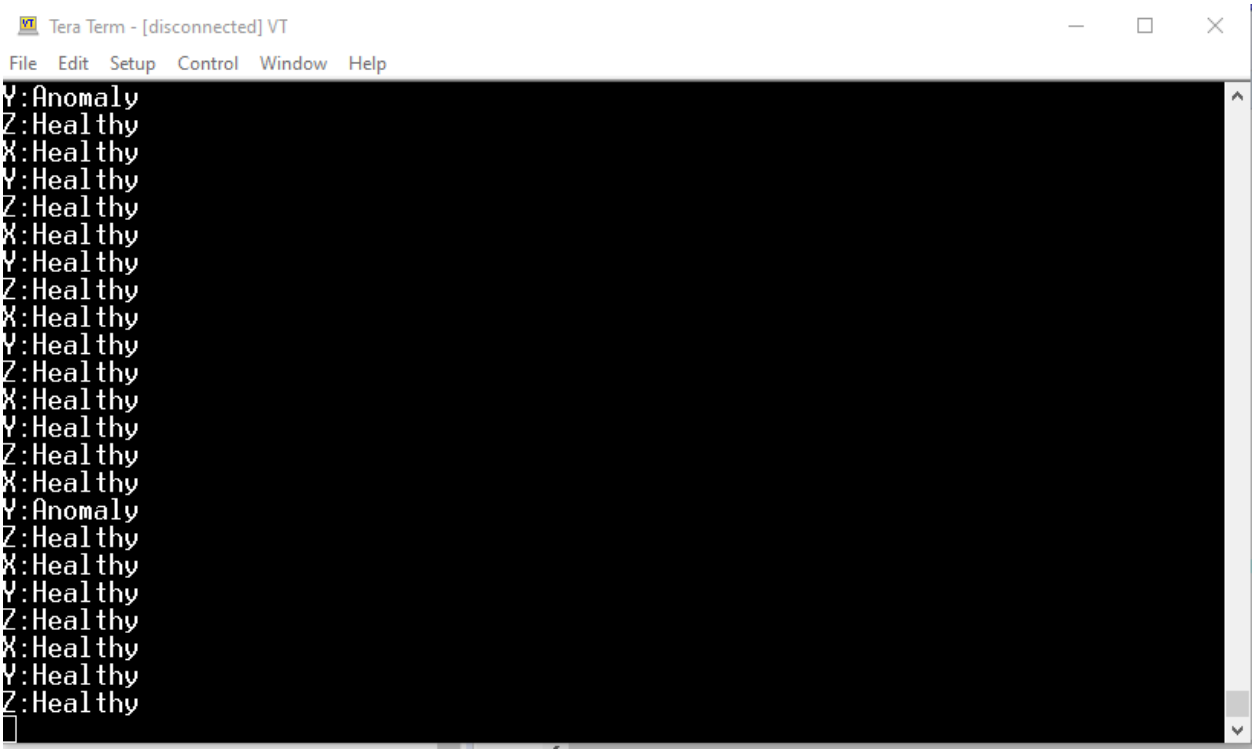
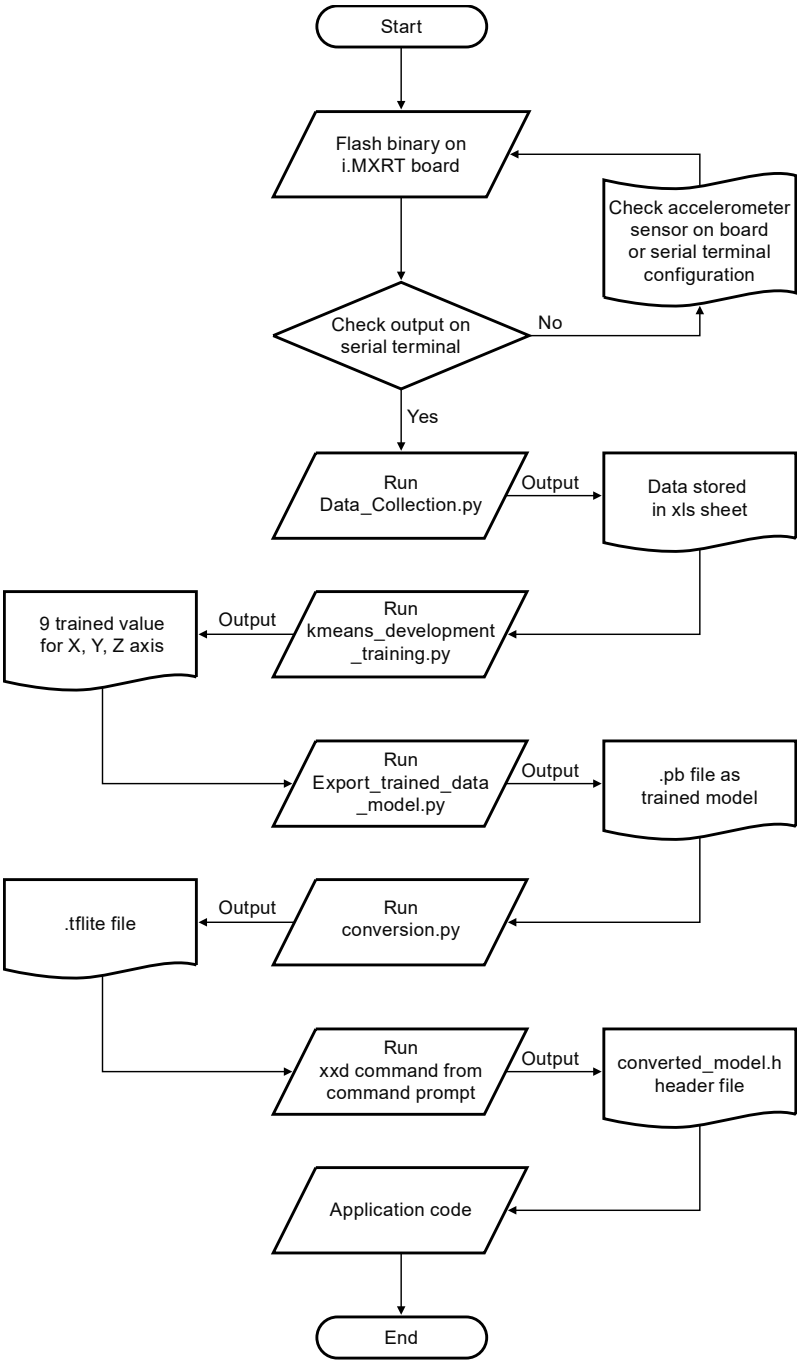


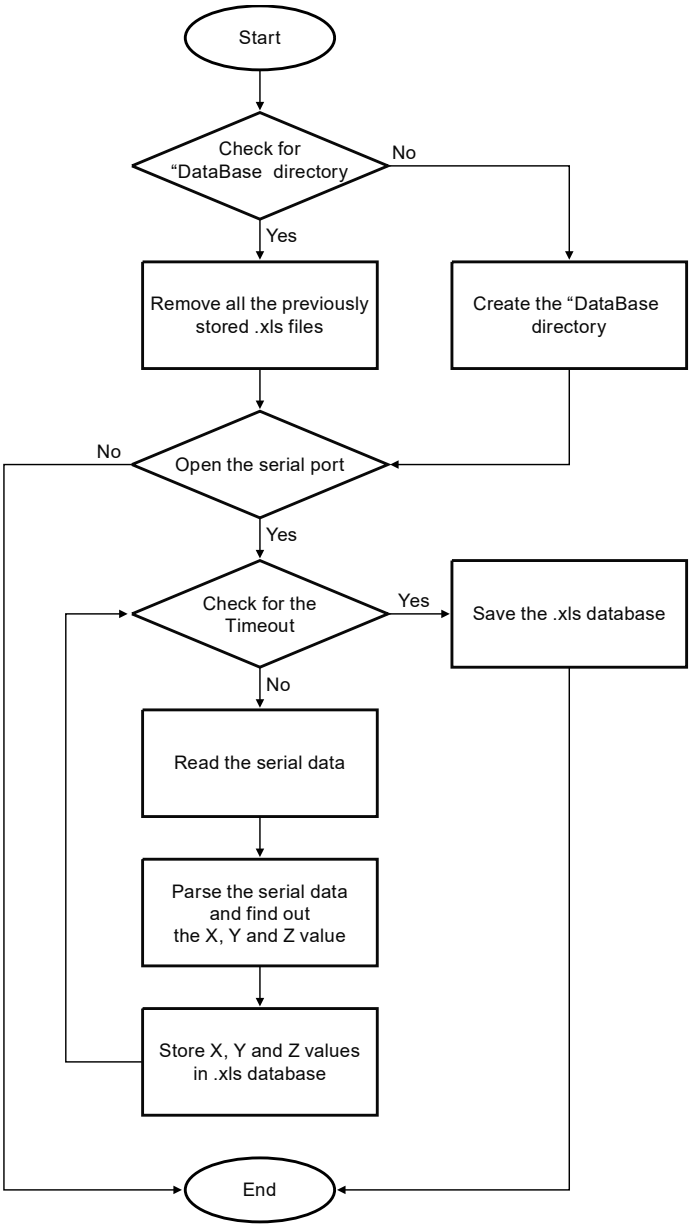
Figure 16. Application output on serial console

A.1.3 Script description flowcharts

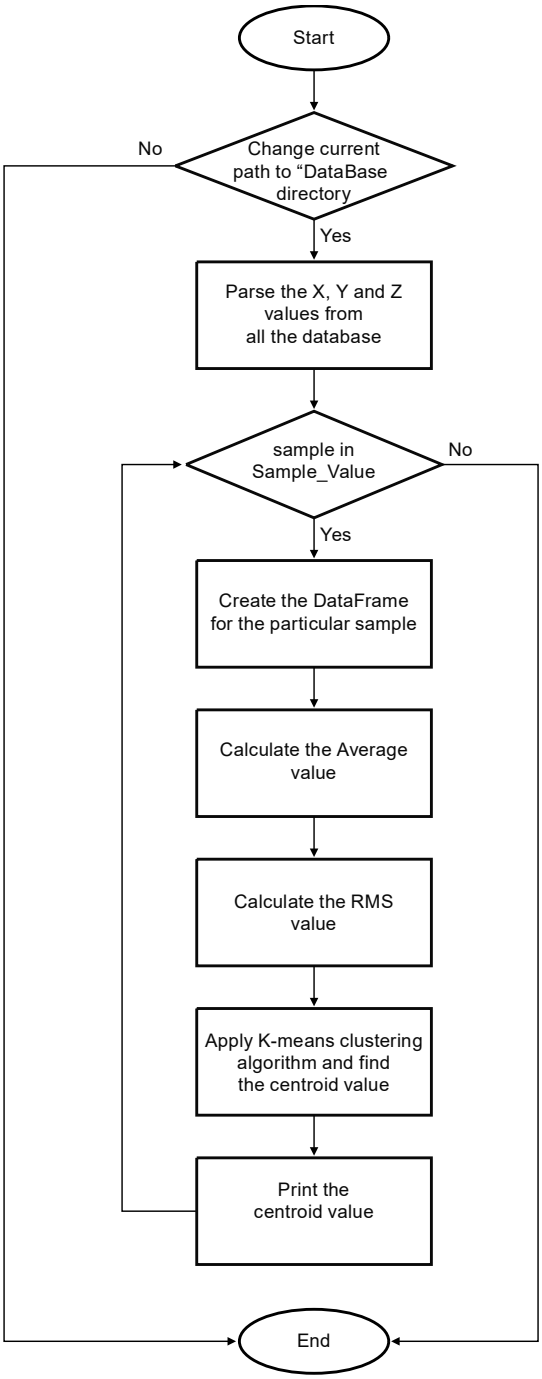
A.1.3.1 Flowchart for level transition diagram for model training and its deployment on edge.



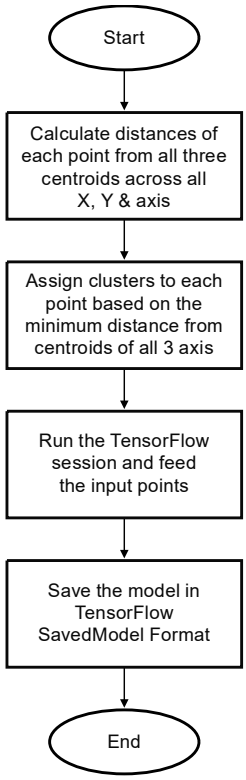
A.1.3.2 Flowchart for execution of Data_Collection.py



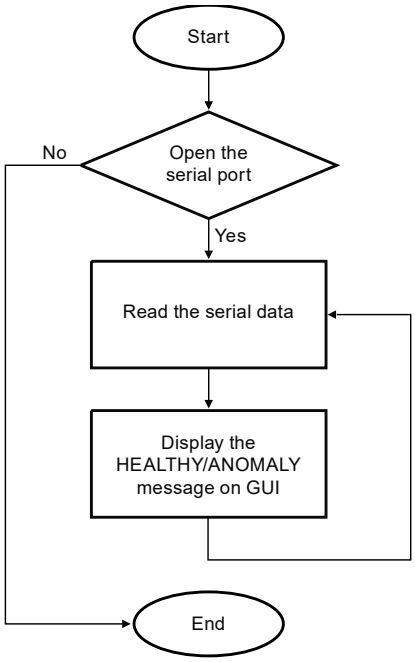
A.1.3.3 Flowchart for execution of Kmeans_Development_Training.py



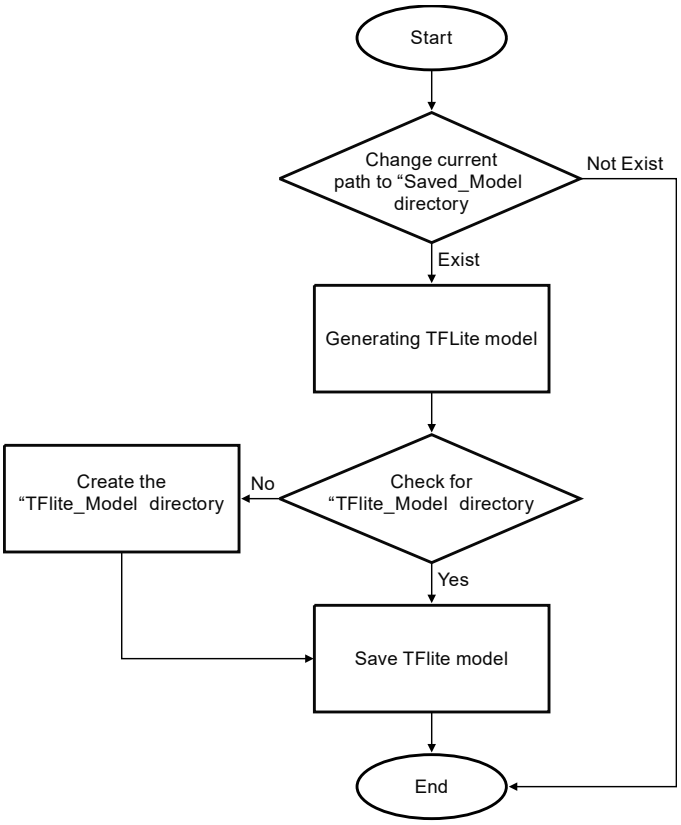
A.1.3.4 Flowchart for execution of Export_Trained_Data_Model.py



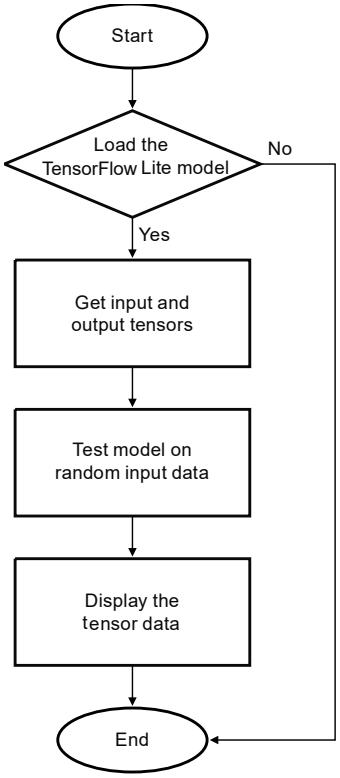
A.1.3.5 Flowchart for execution of Display.py



A.1.3.6 Flowchart for execution of Conversion.py



A.1.3.7 Flowchart for execution of Check_TFlite.py



How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 11/ 2020

Document identifier: AN12766

arm