

Agora | Getting started with Edge Impulse

3 Dec 2021 | v1.0.0



www.embeddedplanet.com

1. **Prerequisites**

NOTE: This tutorial is based off of Edge Impulse's continuous motion recognition tutorial here

<https://docs.edgeimpulse.com/docs/continuous-motion-recognition>

In order to start uploading data from the EP\_AGORA to Edge Impulse, you will need working installations of:

[Mbed CLI](https://os.mbed.com/docs/mbed-os/v6.7/quick-start/build-with-mbed-cli.html) [Edge Impulse Data Forwarder](https://docs.edgeimpulse.com/docs/cli-data-forwarder) Your preferred serial program (TeraTerm, PuTTY, Minicom, etc.)

<https://docs.edgeimpulse.com/docs/cli-data-forwarder>

Firstly, sensor data need to be collected and uploaded to the Edge Impulse cloud for training. The data are collected and sent like this:

Agora ----> Dev Machine running Edge Data Forwarder ----> Edge Impulse Cloud

For example here, datapoints can be collected by reading from the LSM9DS1 accelerometer sensor. Then, after reading from each axis, the data can be formatted and sent over the serial bus to the machine you are using for development.

1. **Setting Up Edge Impulse Data Forwarder**

The Edge Impulse forwarder is expecting individual sensor data to be separated with commas or tabs, with each datapoint being on a new line. The default baud rate of the forwarder is 115200. You can get the Agora to read and forward the data via an mbed program:

1. Use mbed-cli to start a project by running mbed new acquire-data. See mbed-cli's documentation for more information about starting and managing projects.
2. Next, you'll need to add the EP libraries to interface with the sensors. To do this, run mbed add https://github.com/embeddedplanet/ep-oc-mcu/#17fedb4031ede8018d3916a51b3b6d2008dbb6af in the directory of your project.
3. Then, you need to configure the project to use agora specific settings. Copy the following text into mbed\_app.json, replacing the file's existing contents:

{

 "target\_overrides": {

 "\*": {

 "target.features\_add" : ["BOOTLOADER", "STORAGE"],

 "platform.stdio-baud-rate" : 115200,

 "platform.stdio-convert-newlines" : true,

 "platform.stdio-buffered-serial" : true,

 "platform.stdio-flush-at-exit" : true,

 "rtos.main-thread-stack-size" : 5120,

 "mbed-trace.enable" : null,

 "events.shared-stacksize" : 2048,

 "nsapi.default-wifi-security" : "WPA\_WPA2",

 "nsapi.default-wifi-ssid" : "\"SSID\"",

 "nsapi.default-wifi-password" : "\"PASSWORD\"",

 "nsapi.default-cellular-apn" : "\"APN\"",

 "nsapi.default-cellular-sim-pin" : null,

 "nsapi.default-cellular-username" : null,

 "nsapi.default-cellular-password" : null

 },

 "EP\_AGORA": {

 "target.features\_remove" : ["BLE", "CRYPTOCELL310"],

 "target.header\_offset" : "0x10000",

 "target.app\_offset" : "0x10400",

 "target.components\_remove" : ["QSPIF"],

 "target.components\_add" : ["SPIF"],

 "target.macros\_remove" : ["MBEDTLS\_CONFIG\_HW\_SUPPORT"],

 "storage\_filesystem.internal\_base\_address" : "(MBED\_ROM\_START + MBED\_BOOTLOADER\_SIZE)",

 "storage\_filesystem.rbp\_internal\_size" : "(32\*1024)",

 "storage.storage\_type" : "FILESYSTEM",

 "storage\_filesystem.filesystem" : "LITTLE",

 "storage\_filesystem.blockdevice" : "SPIF",

 "storage\_filesystem.external\_base\_address" : "(0x0)",

 "storage\_filesystem.external\_size" : "(1024\*1024\*1)",

 "drivers.uart-serial-rxbuf-size" : 1024,

 "drivers.uart-serial-txbuf-size" : 1024,

 "lwip.ipv4-enabled" : true,

 "lwip.ipv6-enabled" : true,

 "lwip.ppp-enabled" : true,

 "lwip.tcp-enabled" : true,

 "lwip.ethernet-enabled" : false,

 "lwip.mem-size" : 22000,

 "nsapi.dns-response-wait-time" : 30000

 }

 },

 "config": {

 "bootloader-size": {

 "help" : "Helper macro to enable calculation of rom regions. target.header\_offset and target.app\_offset still needs to be calculated manually, though.",

 "value" : "(32\*1024)",

 "macro\_name": "MBED\_BOOTLOADER\_SIZE"

 }

 }

}

1. Here is the code that will be forwarding the data, which you can copy into main.cpp:

#**include** "mbed.h"

#**include** "LSM9DS1.h"

#**define** G\_TO\_MS2 9.8067

**static** **const** **uint8\_t** LSM9DS1\_ACCEL\_GYRO\_ADDRESS = 0x6A << 1;

**static** **const** **uint8\_t** LSM9DS1\_MAG\_ADDRESS = 0x1C << 1;

DigitalOut **sensor\_power\_enable**(PIN\_NAME\_SENSOR\_POWER\_ENABLE);

I2C **i2c**(PIN\_NAME\_SDA, PIN\_NAME\_SCL);

LSM9DS1 **lsm9ds1**(i2c, LSM9DS1\_ACCEL\_GYRO\_ADDRESS, LSM9DS1\_MAG\_ADDRESS);

**int** **main**()

{

 sensor\_power\_enable = 1;

 **if** (lsm9ds1.begin())

 {

 printf("LSM9DS1 online\n");

 lsm9ds1.calibrate();

 } **else**

 {

 printf("ERROR: LSM9DS1 offline!\n");

 }

 **while**(1)

 {

 lsm9ds1.readAccel();

 printf("%f,%f,%f\n",lsm9ds1.calcAccel(lsm9ds1.ax) \* G\_TO\_MS2, lsm9ds1.calcAccel(lsm9ds1.ay) \* G\_TO\_MS2, lsm9ds1.calcAccel(lsm9ds1.az) \* G\_TO\_MS2);

 ThisThread::sleep\_for(1);

 }

 **return** 0;

}

1. Lastly run a build with mbed compile -t GCC\_ARM -m EP\_AGORA and then copy the build output to the agora. On Linux, you will need to check how the device enumerated, which can be done by running dmesg right after you connect the agora. Near the bottom of the output, you should see something like this:



"sda" corresponds to the mass storage enumeration, and ttyACM0 corresponds to the serial port. Replace the following instances of "sda" and "ttyACM0" with the values from your dmesg output. To upload the build to the agora, mount the mass storage device: sudo mount /dev/sda /mnt and then copy the program over: sudo cp BUILD/EP\_AGORA/GCC\_ARM/send-data.hex /mnt && sync. Then, set up your serial program (such as TeraTerm) to communicate with the device "/dev/ttyACM0" with a baud rate of 115200 and no flow control.

After uploading the sampling code to the Agora, you can run the edge-impulse-data-forwarder and follow its prompts to provide your login credentials. The data forwarder is a nodeJS application that opens a websocket with the Edge Impulse cloud. This lets you control and label sampling from their web application instead of having to deal with files of sampled data on your local machine.

1. **Collecting Data**

Now, you can more closely follow Edge Impulse's [tutorial](https://docs.edgeimpulse.com/docs/continuous-motion-recognition) at Step 2: "Collecting your first data."

<https://docs.edgeimpulse.com/docs/continuous-motion-recognition>

At step 6, select "C++ library" from the deployment menu and then click "Build." Then, in the parent directory of your project, you'll need to import Edge Impulse's example standalone project, which contains the boilerplate code needed to use the classifier you made. After that, you will need to follow a similar procedure to the data-forwarder to use agora-specific sensors and settings. All of that can be encapsulated in these steps:

1. Run mbed import https://github.com/edgeimpulse/example-standalone-inferencing-mbed
2. Extract the .zip file you downloaded from step 6 of Edge Impulse's tutorial in the new project directory.
3. Run mbed add https://github.com/embeddedplanet/ep-oc-mcu/#17fedb4031ede8018d3916a51b3b6d2008dbb6af
4. Copy the mbed\_app.json of the acquire-data project to this one. You will need to add a line to the "\*" block of "target\_overrides," which is: "target.printf\_lib": "std"
5. Now, copy this code to source/main.cpp:

#**include** "mbed.h"

#**include** "LSM9DS1.h"

#**include** "ei\_run\_classifier.h"

#**define** G\_TO\_MS2 9.8067

**static** **const** **uint8\_t** LSM9DS1\_ACCEL\_GYRO\_ADDRESS = 0x6A << 1;

**static** **const** **uint8\_t** LSM9DS1\_MAG\_ADDRESS = 0x1C << 1;

DigitalOut **sensor\_power\_enable**(PIN\_NAME\_SENSOR\_POWER\_ENABLE);

I2C **i2c**(PIN\_NAME\_SDA, PIN\_NAME\_SCL);

LSM9DS1 **lsm9ds1**(i2c, LSM9DS1\_ACCEL\_GYRO\_ADDRESS, LSM9DS1\_MAG\_ADDRESS);

**static** **int64\_t** sampling\_freq = EI\_CLASSIFIER\_FREQUENCY; // in Hz.

**static** **int64\_t** time\_between\_samples\_us = (1000000 / (sampling\_freq - 1));

// to classify 1 frame of data you need EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE values

**static** **float** features[EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE];

// set baud rate of serial port to 115200

**static** BufferedSerial **serial\_port**(USBTX, USBRX, 115200);

FileHandle \***mbed::mbed\_override\_console**(**int** fd) {

 **return** &serial\_port;

}

**int** **main**()

{

 Timer t;

 t.start();

 sensor\_power\_enable = 1;

 **if** (lsm9ds1.begin())

 {

 printf("LSM9DS1 online\n");

 lsm9ds1.calibrate();

 } **else**

 {

 printf("ERROR: LSM9DS1 offline!\n");

 }

 **while** (1) {

 // fill the features array

 **for** (**size\_t** ix = 0; ix < EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE; ix += EI\_CLASSIFIER\_RAW\_SAMPLES\_PER\_FRAME) {

 **int64\_t** next\_tick = t.read\_us() + time\_between\_samples\_us;

 lsm9ds1.readAccel();

 // copy accelerometer data into the features array

 features[ix + 0] = lsm9ds1.calcAccel(lsm9ds1.ax) \* G\_TO\_MS2;

 features[ix + 1] = lsm9ds1.calcAccel(lsm9ds1.ay) \* G\_TO\_MS2;

 features[ix + 2] = lsm9ds1.calcAccel(lsm9ds1.az) \* G\_TO\_MS2;

 **while** (t.read\_us() < next\_tick) {

 /\* busy loop \*/

 }

 }

 // frame full? then classify

 **ei\_impulse\_result\_t** result = { 0 };

 // create signal from features frame

 **signal\_t** signal;

 numpy::signal\_from\_buffer(features, EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE, &signal);

 // run classifier

 EI\_IMPULSE\_ERROR res = run\_classifier(&signal, &result, false);

 ei\_printf("run\_classifier returned: %d\n", res);

 **if** (res != 0) **return** 1;

 // print predictions

 ei\_printf("Predictions (DSP: %d ms., Classification: %d ms., Anomaly: %d ms.): \n",

 result.timing.dsp, result.timing.classification, result.timing.anomaly);

 // print the predictions

 **for** (**size\_t** ix = 0; ix < EI\_CLASSIFIER\_LABEL\_COUNT; ix++) {

 ei\_printf("%s:\t%.5f\n", result.classification[ix].label, result.classification[ix].value);

 }

 #**if** EI\_CLASSIFIER\_HAS\_ANOMALY == 1

 ei\_printf("anomaly:\t%.3f\n", result.anomaly);

 #**endif**

 }

}

1. Lastly, compile and run on your device!

$ mbed **compile** -t GCC\_ARM -m EP\_AGORA

$ sudo cp BUILD/EP\_AGORA/GCC\_ARM/example-standalone-inferencing-mbed.hex /mnt && sync

Use your favorite serial port reading program to see the classifier output!