Clasificación de Movimiento:

Arduino Tiny ML Kit

Juan Camilo Giraldo Londoño
Karen Sofia Bustamante Villota


Hand Module (with Motion Sensor) for Hand Interaction
Self Initiated, Active, Repetitive Training
Augmented Performance Feedback
3 Motion Sensors: 2 on the arm, 1 on the chest

Clasificación movimiento

Ejercer puño

Flexión antebrazo

Levantar brazo

MUESTREO

- 3 movimientos.
- 10 ejercicios por movimiento.
- 119 muestras por ejercicio.

```
#include <Arduino_LSM9DS1.h>

const float accelerationThreshold = 2.5;
const int numSamples = 119;

int samplesRead = numSamples;
```

1

```
while (samplesRead == numSamples) {
  if (IMU.accelerationAvailable()) {
    // read the acceleration data
    IMU.readAcceleration(aX, aY, aZ);

    // sum up the absolutes
    float aSum = fabs(aX) + fabs(aY) + fabs(aZ);

    // check if it's above the threshold
    if (aSum >= accelerationThreshold) {
      // reset the sample read count
      samplesRead = 0;
      break;
    }
  }
```

2

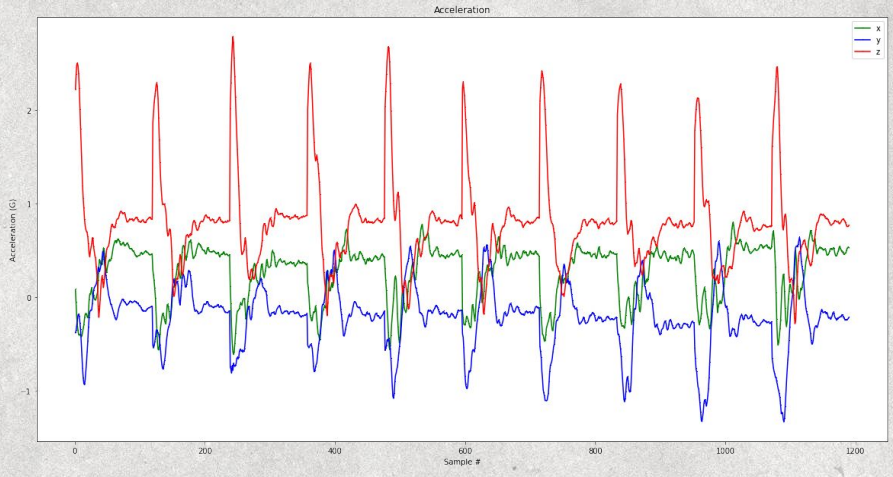
```
while (samplesRead < numSamples) {
  // check if both new acceleration and gyroscope data is
  // available
  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()) {
    // read the acceleration and gyroscope data
    IMU.readAcceleration(aX, aY, aZ);
    IMU.readGyroscope(gX, gY, gZ);

    samplesRead++;
```

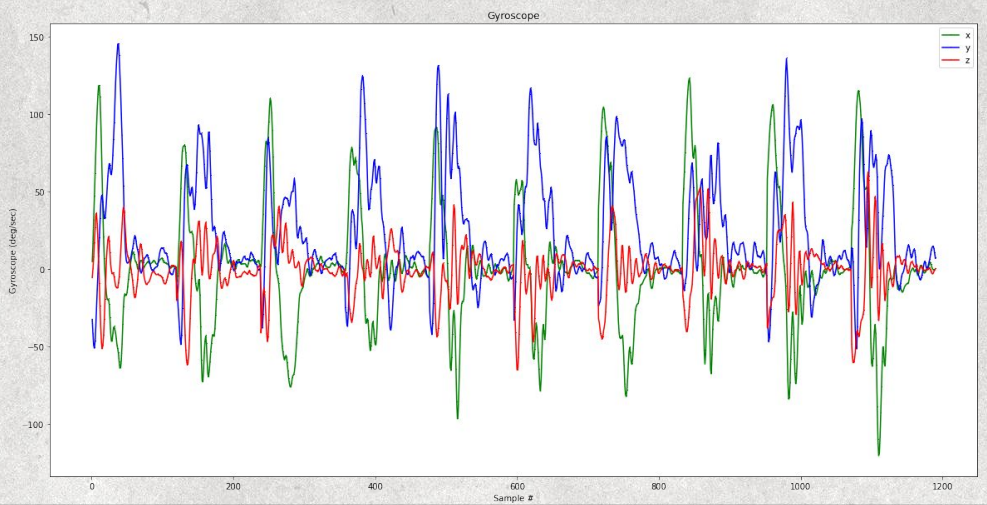
3

ARDUINO

ENTRENAMIENTO



Movimiento:
Levantar brazo



Normalización de los datos de entrada

```
# the list of gestures that data is available for
GESTURES = [
    "punch",
    "flex",
    "up",
]

SAMPLES_PER_GESTURE = 119

NUM_GESTURES = len(GESTURES)

# create a one-hot encoded matrix that is used in the output
ONE_HOT_ENCODED_GESTURES = np.eye(NUM_GESTURES)

inputs = []
outputs = []
```

1

```
# read each csv file and push an input and output
for gesture_index in range(NUM_GESTURES):
  gesture = GESTURES[gesture_index]
  print(f"Processing index {gesture_index} for gesture '{gesture}'.")

  output = ONE_HOT_ENCODED_GESTURES[gesture_index]

  df = pd.read_csv("/content/" + gesture + ".csv")

  # calculate the number of gesture recordings in the file
  num_recordings = int(df.shape[0] / SAMPLES_PER_GESTURE)
```

2

```
  for i in range(num_recordings):
    tensor = []
    for j in range(SAMPLES_PER_GESTURE):
      index = i * SAMPLES_PER_GESTURE + j
      # normalize the input data, between 0 to 1:
      # - acceleration is between: -4 to +4
      # - gyroscope is between: -2000 to +2000
      tensor += [
          (df['aX'][index] + 4) / 8,
          (df['aY'][index] + 4) / 8,
          (df['aZ'][index] + 4) / 8,
          (df['gX'][index] + 2000) / 4000,
          (df['gY'][index] + 2000) / 4000,
          (df['gZ'][index] + 2000) / 4000
      ]

    inputs.append(tensor)
    outputs.append(output)

# convert the list to numpy array
inputs = np.array(inputs)
outputs = np.array(outputs)
```

3

```
um_inputs = len(inputs)
randomize = np.arange(num_inputs)
np.random.shuffle(randomize)

# Swap the consecutive indexes (0, 1, 2, etc) with the randomized indexes
inputs = inputs[randomize]
outputs = outputs[randomize]

# Split the recordings (group of samples) into three sets: training, testing and validation
TRAIN_SPLIT = int(0.6 * num_inputs)
TEST_SPLIT = int(0.2 * num_inputs + TRAIN_SPLIT)

inputs_train, inputs_test, inputs_validate = np.split(inputs, [TRAIN_SPLIT, TEST_SPLIT])
outputs_train, outputs_test, outputs_validate = np.split(outputs, [TRAIN_SPLIT, TEST_SPLIT])
```

Arquitectura del modelo

```
# build the model and train it
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(50, activation='relu')) # relu is used for performance
model.add(tf.keras.layers.Dense(15, activation='relu'))
model.add(tf.keras.layers.Dense(NUM_GESTURES, activation='softmax')) # softmax is used, because we only expect one gesture to occur per input
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
history = model.fit(inputs_train, outputs_train, epochs=600, batch_size=1, validation_data=(inputs_validate, outputs_validate))
```

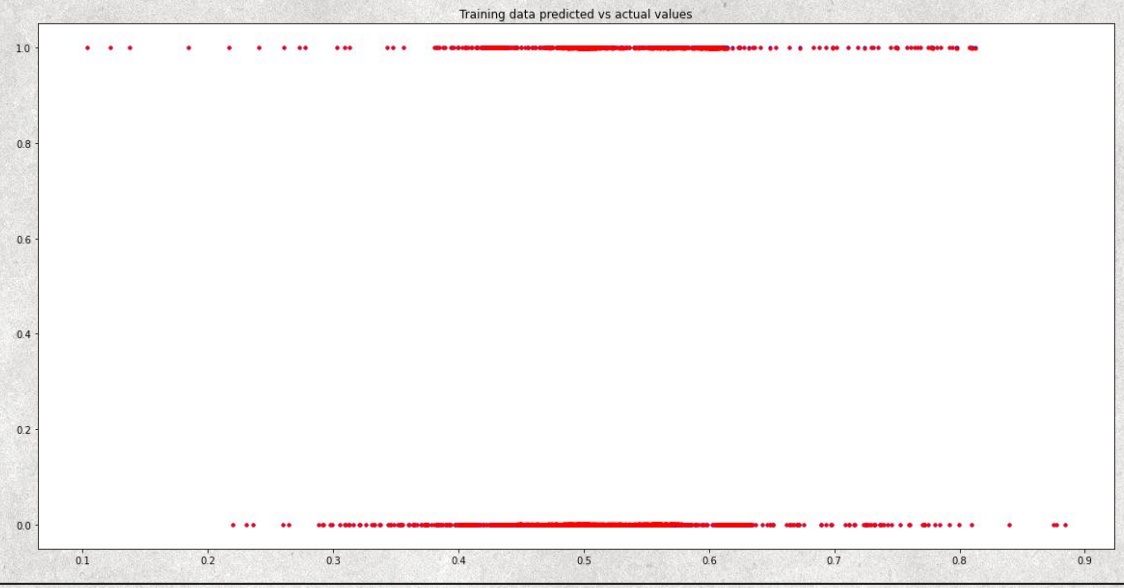
```
Epoch 600/600
18/18 [==============================] - 0s 6ms/step - loss: 5.9828e-07 - mae: 3.8299e-04 - val_loss: 9.4761e-07 - val_mae: 4.5839e-04
```

Resultados

```
# use the model to predict the test inputs
predictions = model.predict(inputs_test)

# print the predictions and the expected ouputs
print("predictions =\n", np.round(predictions, decimals=3))
print("actual =\n", outputs_test)

# Plot the predictions along with to the test data
plt.clf()
plt.title('Training data predicted vs actual values')
plt.plot(inputs_test, outputs_test, 'b.', label='Actual')
plt.plot(inputs_test, predictions, 'r.', label='Predicted')
plt.show()
```



```
# Convert the model to the TensorFlow Lite format without quantization
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model to disk
open("gesture_model.tflite", "wb").write(tflite_model)

import os
basic_model_size = os.path.getsize("gesture_model.tflite")
print("Model is %d bytes" % basic_model_size)
```

```
!echo "const unsigned char model[] = {" > /content/model.h
!cat gesture_model.tflite | xxd -i      >> /content/model.h
!echo "};"                              >> /content/model.h

import os
model_h_size = os.path.getsize("model.h")
print(f"Header file, model.h, is {model_h_size:,} bytes.")
print("\nOpen the side panel (refresh if needed). Double click model.h to download the file.")
```

Despliegue

```
#include <Arduino_LSM9DS1.h>

#include <TensorFlowLite.h>
#include <tensorflow/lite/micro/all_ops_resolver.h>
#include <tensorflow/lite/micro/micro_error_reporter.h>
#include <tensorflow/lite/micro/micro_interpreter.h>
#include <tensorflow/lite/schema/schema_generated.h>
#include <tensorflow/lite/version.h>

#include "model.h"
```

```
// get the TFL representation of the model byte array
tflModel = tflite::GetModel(model);
if (tflModel->version() != TFLITE_SCHEMA_VERSION) {
  Serial.println("Model schema mismatch!");
  while (1);
}

// Create an interpreter to run the model
tflInterpreter = new tflite::MicroInterpreter(tflModel, tflOpsResolver, tensorArena, tensorArenaSize, &tflErrorReporter);

// Allocate memory for the model's input and output tensors
tflInterpreter->AllocateTensors();

// Get pointers for the model's input and output tensors
tflInputTensor = tflInterpreter->input(0);
tflOutputTensor = tflInterpreter->output(0);
```

```
  check if the all the required samples have been read since
// the last time the significant motion was detected
while (samplesRead < numSamples) {
  // check if new acceleration AND gyroscope data is available
  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()) {
    // read the acceleration and gyroscope data
    IMU.readAcceleration(aX, aY, aZ);
    IMU.readGyroscope(gX, gY, gZ);

    // normalize the IMU data between 0 to 1 and store in the model's
    // input tensor
    tflInputTensor->data.f[samplesRead * 6 + 0] = (aX + 4.0) / 8.0;
    tflInputTensor->data.f[samplesRead * 6 + 1] = (aY + 4.0) / 8.0;
    tflInputTensor->data.f[samplesRead * 6 + 2] = (aZ + 4.0) / 8.0;
    tflInputTensor->data.f[samplesRead * 6 + 3] = (gX + 2000.0) / 4000.0;
    tflInputTensor->data.f[samplesRead * 6 + 4] = (gY + 2000.0) / 4000.0;
    tflInputTensor->data.f[samplesRead * 6 + 5] = (gZ + 2000.0) / 4000.0;

    samplesRead++;
```

Referencias

- Ejercicio realizado en base a Gesture recognition tutorial: Sandeep Mistry - Arduino & Don Coleman - Chariot Solutions.

¡Muchas gracias por su atención!