UNIVERSIDAD AUTÓNOMA DE OCCIDENTE
FACULTAD DE INGENIERÍA.
PROCESAMIENTO DE DATOS SECUENCIALES CON DEEP LEARNING
ESPECIALIZACIÓN EN INTELIGENCIA ARTIFICIAL
PROFESOR: JESÚS ALFONSO LÓPEZ

IMPLEMENTACIÓN DE MODELOS DE INTELIGENCIA ARTIFICIAL USANDO UNA TARJETA DE DESARROLLO

Integrantes: *Diana Delgado Paz*
Edison Orozco
Sergio Duvan Mendoza Rojas

Introducción:

La idea principal de este laboratorio es ofrecer una alternativa de manipulación al robot *Escornabot***,** mediante 4 simples movimientos: inclinando la mano hacia arriba (reversa), girando la mano (avanzar), y en esa misma posición al detectar cambio de posición a la derecha e izquierda para sus respectivos movimientos.
Se utilizará la IMU (Unidad de Medición Inercial) presente en la placa Arduino Nano 33 BLE Sense para controlar el robot, haciendo uso de los 9 sensores que posee (Aceleración, Giroscopio, Campo magnético) para tener más información relevante.

Objetivos:

El principal objetivo es generar un archivo en tensorflow lite para microcontroladores, que trabaje con los datos entregados por los sensores sin que haya mucho procesamiento de por medio para disminuir el consumo de recursos del embebido a usar, siendo más fácil de usar al momento de identificar los gestos realizados.

Descripción del proyecto:

El proyecto se divide en dos partes, una es haciendo uso de la plataforma Edge impulse, el segundo es con la implementación en Tensorflow, a continuación se describe el proceso realizado en cada una.

1. *Plataforma Edge Impulse*:

   El proyecto en la plataforma edge impulse bajo el nombre de Movimientos robot con arduino ble 33, permite visualizar la adquisición de datos y su posterior entrenamiento mediante una arquitectura compuesta por dos capas convolucionales de 1 dimensión, seguida de un max pooling y finalizando con un clasificador Softmax.

   La fuente de los datos es el mismo arduino nano ble 33, se conecta mediante el uso del cliente de edge impulse en linux con el comando '*edge-impulse-daemon*'.

Para la adquisición de los datos, se generan ventanas de 1 segundo, en el que se reconocerá un movimiento específico, tal como se muestra a continuación

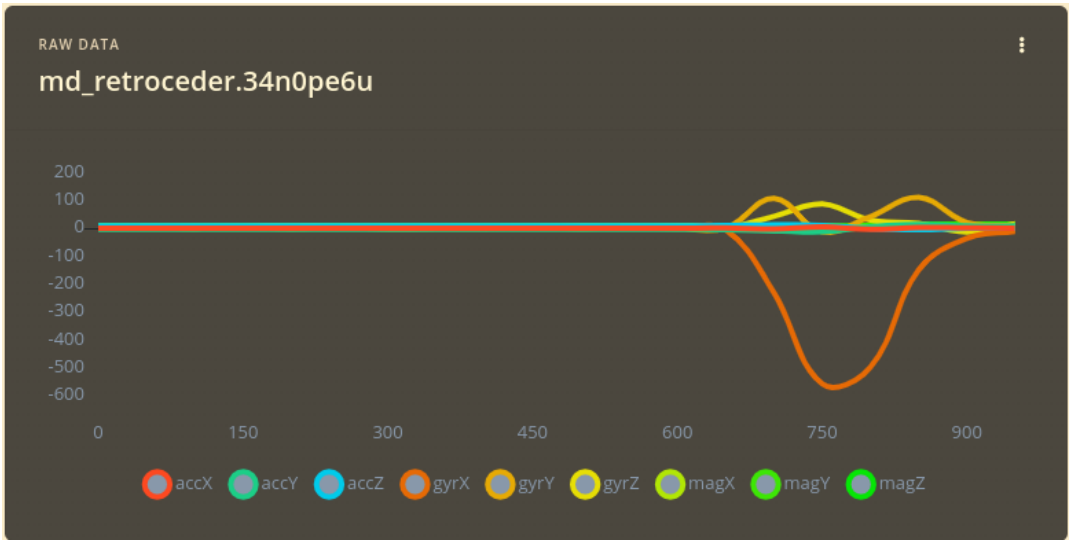


Imagen 1. Adquisición de datos usando una ventana de 1 segundo.

Al implementar los 9 sensores del que dispone el arduino, se puede diferenciar más el tipo de movimiento realizado, con el propósito de identificar las curvas.

Los movimientos que se realizaron fueron capturados en las siguientes poses.

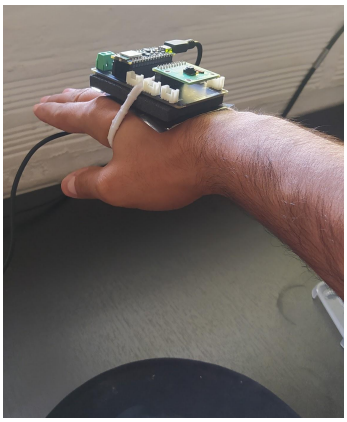
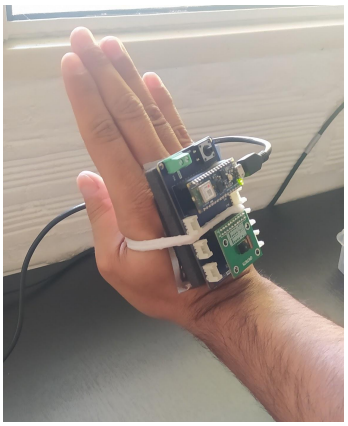
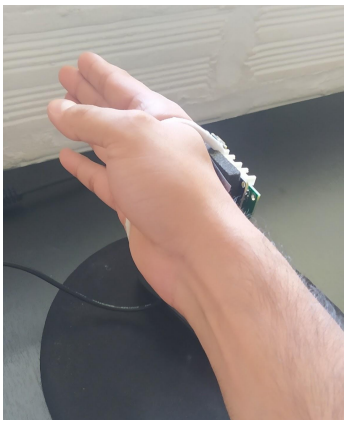
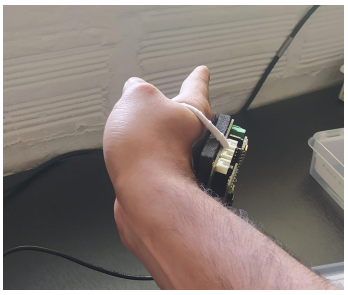
 Mano quieta	 reversa	 avanzar
 giro izquierda	 giro derecha	

Imagen 2. Estimación de poses.

Después de esto la red creada se hace mediante código, ya que al usar los bloques de la plataforma edge impulse, generaba error de entradas.

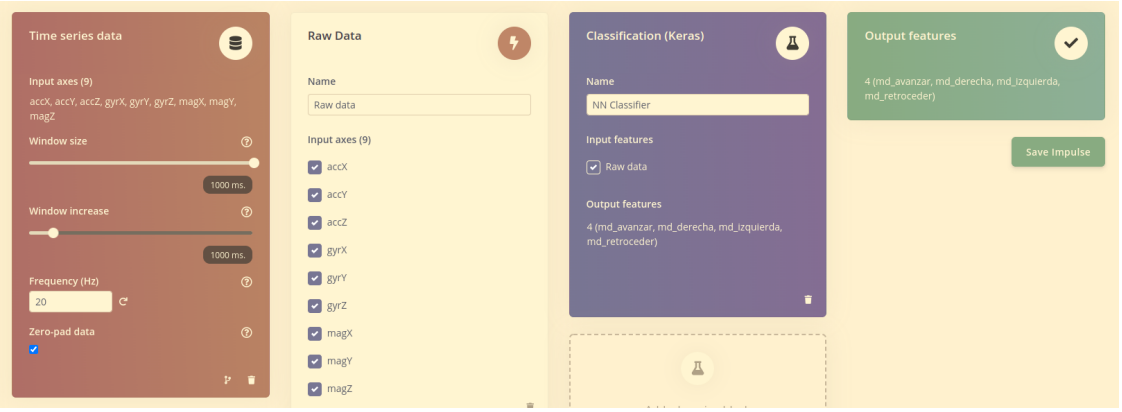


Imagen 3. Diseñando arquitectura a usar en edge impulse

En la entrada a las capas Conv1D se indica el tamaño de los datos, que lee el propio edge impulse.

```
# model architecture
model = Sequential()
model.add(Conv1D(16, kernel_size=3, activation='relu', padding='same', input_shape=(180,1)))
model.add(Conv1D(32, kernel_size=3, activation='relu', padding='same', input_shape=(180,1)))
model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
model.add(Flatten())
model.add(Dense(classes, activation='softmax', name='y_pred'))
```

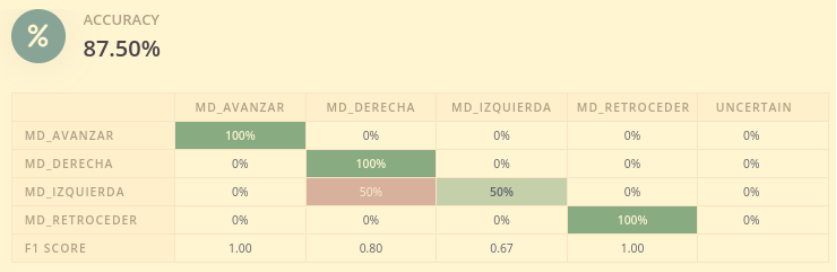
Después se comprueba la matriz de confusión tanto del entrenamiento como de la validación

para el entrenamiento se tiene una exactitud del 100%, lo que nos da a entender que la arquitectura se amoldo muy bien a estos datos.

Confusion matrix (validation set)

	MD_AVANZAR	MD_DERECHA	MD_IZQUIERDA	MD_RETROCEDER
MD_AVANZAR	100%	0%	0%	0%
MD_DERECHA	0%	100%	0%	0%
MD_IZQUIERDA	0%	0%	100%	0%
MD_RETROCEDER	0%	0%	0%	100%
F1 SCORE	1.00	1.00	1.00	1.00

Al realizar la validación del modelo, se ve que el modelo generaliza bien, y se puede ver que en el movimiento de Izquierda, presenta un solo fallo.



Finalmente procedemos a realizar un despliegue para el microcontrolador arduino, en cuyo caso nos entrega un archivo *'flash_linux.sh*' para programarlo y se puede observar los resultados de la clasificación usando el comando '*edge-impulse-run-impulse*'

Algunos resultados son:

Predicción de retroceder:

```
Predictions (DSP: 0 ms., Classification: 6 ms., Anomaly: 0 ms.):
    md_avanzar: 0.04688
    md_derecha: 0.02344
    md_izquierda: 0.04688
    md_retroceder: 0.88672
```

Predicción de giro derecha:

```
Predictions (DSP: 0 ms., Classification: 6 ms., Anomaly: 0 ms.):
    md_avanzar: 0.03906
    md_derecha: 0.72266
    md_izquierda: 0.23828
    md_retroceder: 0.00391

Predictions (DSP: 0 ms., Classification: 6 ms., Anomaly: 0 ms.):
    md_avanzar: 0.12891
    md_derecha: 0.56641
    md_izquierda: 0.27344
    md_retroceder: 0.03125
```

Se nota que para algunos movimientos se presenta un error en la clasificación, pero es posible definir el tipo de movimiento.

2. *Tensorflow en Local / Google Collaboratory*:

Para el procedimiento con el framework tensorflow, se realiza los siguientes pasos, se agrega también el acceso al repositorio en github [Movimiento Robot con arduino ble 33](#).

Para la adquisición de datos, se usó la plataforma edge impulse, pero para usarse en el notebook se usaron los siguientes comandos, debido a que se descargaban archivos en formato ‘cbor’:

Se instala la librería cbor

```
pip3 install cbor2
```

Se abre el archivo con el comando:

```
from cbor2 import load as cb_load
with open(RutaFile, 'rb') as fp:
    obj = cb_load(fp,)

# El obj es un diccionario, por lo que después de explorar el contenido
# Se puede cargar como un dataframe.

dataframe = pd.DataFrame(obj['payload']['values'],index=index_list,columns=column_list)
```

Y ya con este dataframe se puede manipular más fácilmente los datos, concatenarlos, y normalizar:

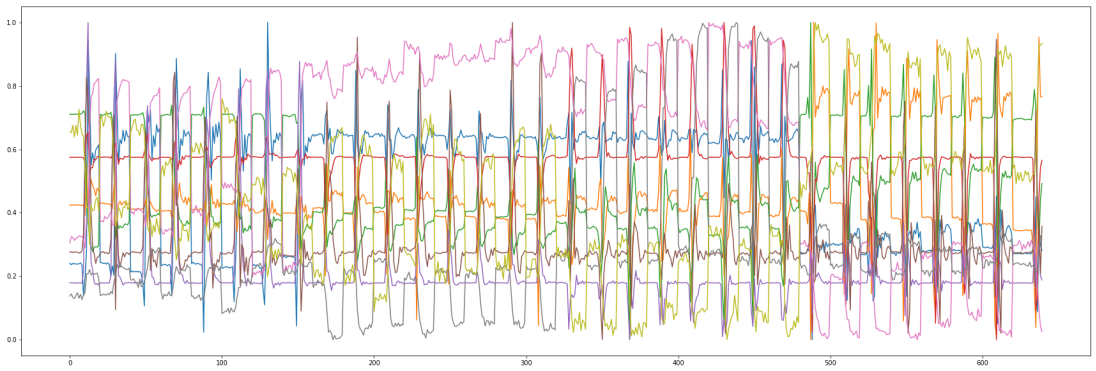


Imagen 4. Dataset entrenamiento concatenado y normalizado

Como se nota, se deben realizar ventanas y sus respectivas etiquetas, las ventanas en este caso representan un segundo en que las muestras fueron tomadas a 20 Hz por usar el sensor de Campo magnético

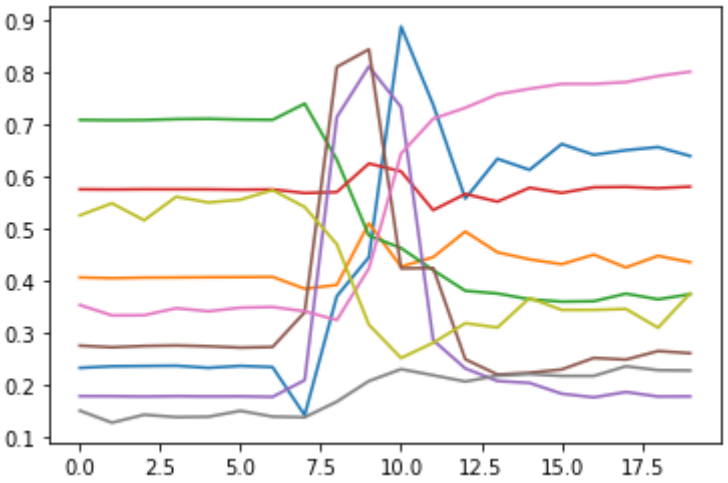


Imagen 5. Dataset separado en ventanas de a 20 datos.

Para la arquitectura a utilizar, se realizaron pruebas con varias, entre ellas una con solo Convolución 1D, Simple RNN y LSTM, para destacar en este informe, la arquitectura Conv1D no estaba ejecutándose bien con la librería tflite de arduino, por lo que se decidió usar una Conv2D como se ve a continuación

```
#Definición del modelo
modelo = Sequential()
# modelo.add(Conv1D(16,3, activation="relu", padding="same", input_shape=(20,9),))
# modelo.add(Conv1D(32,3, activation="relu", padding="same", input_shape=(20,9)))
# modelo.add(MaxPooling1D(pool_size=2, strides=2,padding="same"))
modelo.add(Reshape((1,20,9),input_shape=(20,9)))
modelo.add(Conv2D(16,(3,1), activation="relu", padding="same"))
modelo.add(Conv2D(32,(3,1), activation="relu", padding="same"))
modelo.add(MaxPooling2D(pool_size=(2,1), strides=(2,1),padding="same"))
modelo.add(Flatten())
modelo.add(Dense(4,activation="softmax"))

modelo.summary()

keras.utils.plot_model(modelo, to_file='model_HAR_conv1D.png', show_shapes=True, show_layer_names=True)
```

También, para destacar, aunque los gráficos de pérdida entrenamiento y testeo, se ajustan adecuadamente, la arquitectura con SRNN, esta presenta mayor error al momento de predecir, y esto se refleja a su vez en la matriz de confusión, siendo la peor arquitectura a utilizar.

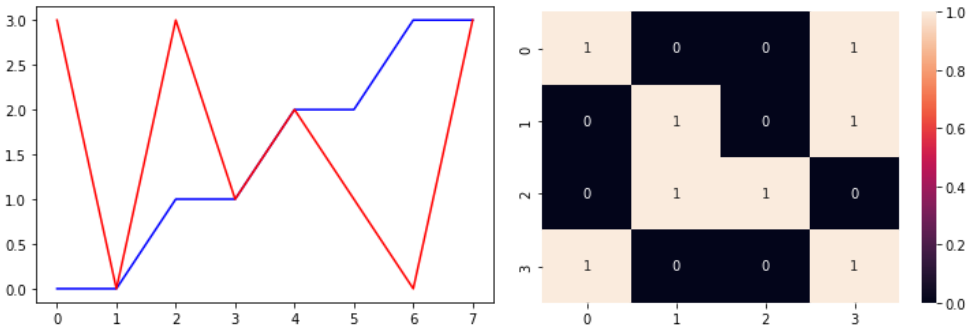


Imagen 6. Predicción de arquitectura SRNN (Izquierda), Matriz confusión (derecha)

Después de convertidos todos los modelos a TFLite, se observa que la pérdida en los diferentes modelos mantiene su relación, siendo SRNN la peor, y en un caso extraño, la LSTM se vuelve peor al cuantizar

Tabla 1. Pérdida de cada modelo y su conversiones.

	Loss/MSE		
Model	**Conv1D**	**SRNN**	**LSTM**
Tensor Flow	0.0001	0.1394	0.0019
Tensor Flow Lite	0.0001	0.1395	0.0019
TF Lite Quantized	0.0230	0.1613	2.0167

Finalmente se genera el modelo cuantizado con el convertidor 'xxd' para usarse como una librería de arduino.

```
# Convert to a C source file, i.e, a TensorFlow Lite for Microcontrollers model
!xxd -i {MODEL_TFLITE_CONV1D} > {MODEL_TFLITE_MICRO_CONV1D}
# Update variable names
REPLACE_TEXT_CONV1D = MODEL_TFLITE_CONV1D.replace('/', '_').replace('.', '_')
!sed -i 's/'{REPLACE_TEXT_CONV1D}'/model/g' {MODEL_TFLITE_MICRO_CONV1D}
```

```
# Print the C source file
!cat $MODEL_TFLITE_MICRO_CONV1D
```

```
0x64, 0x65, 0x6e, 0x73, 0x65, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
0x04, 0x00, 0x00, 0x00, 0x26, 0xe9, 0xff, 0xff, 0x04, 0x00, 0x00, 0x00,
0x0d, 0x00, 0x00, 0x00, 0x72, 0x65, 0x73, 0x68, 0x61, 0x70, 0x65, 0x5f,
0x69, 0x6e, 0x70, 0x75, 0x74, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00,
0x34, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0xdc, 0xff, 0xff, 0xff,
0x19, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x13, 0x00, 0x00, 0x00,
0x43, 0x4f, 0x4e, 0x56, 0x45, 0x52, 0x53, 0x49, 0x4f, 0x4e, 0x5f, 0x4d,
0x45, 0x54, 0x41, 0x44, 0x41, 0x54, 0x41, 0x00, 0x08, 0x00, 0x0c, 0x00,
0x08, 0x00, 0x04, 0x00, 0x08, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00,
0x04, 0x00, 0x00, 0x00, 0x13, 0x00, 0x00, 0x00, 0x6d, 0x69, 0x6e, 0x5f,
0x72, 0x75, 0x6e, 0x74, 0x69, 0x6d, 0x65, 0x5f, 0x76, 0x65, 0x72, 0x73,
0x69, 0x6f, 0x6e, 0x00, 0x1a, 0x00, 0x00, 0x00, 0x34, 0x14, 0x00, 0x00,
```

3. *Arduino Nano BLE 33 Sense*: Para su implementación, se usó el ejemplo de 'IMU_Classifier' del Tutorial de Arduino para TFLite, donde se hicieron las siguientes modificaciones.

Primero se copia los datos hexadecimales anteriormente generados en tensorflow al archivo 'model.h', este se encargará de hacer la inferencia en el arduino.

Se agregan los gestos realizados

```
const char* GESTURES[] = {
  "md_avanzar",
  "md_derecha",
  "md_izquierda",
  "md_retroceder"
};
```

Se procede a hacer la lectura de los tres sensores:

```
if (IMU.accelerationAvailable() &&
    IMU.gyroscopeAvailable() &&
    IMU.magneticFieldAvailable()) {
      // read the acceleration and gyroscope data
      IMU.readAcceleration(aX, aY, aZ);
      IMU.readGyroscope(gX, gY, gZ);
      IMU.readMagneticField(mX, mY, mZ);
```

A su vez, se normalizan los datos

```
// normalize the IMU data between 0 to 1 and store in the model's
// input tensor
tflInputTensor->data.f[samplesRead * 9 + 0] = (aX + 4.0) / 8.0;
tflInputTensor->data.f[samplesRead * 9 + 1] = (aY + 4.0) / 8.0;
tflInputTensor->data.f[samplesRead * 9 + 2] = (aZ + 4.0) / 8.0;
tflInputTensor->data.f[samplesRead * 9 + 3] = (gX + 2000.0) / 4000.0;
tflInputTensor->data.f[samplesRead * 9 + 4] = (gY + 2000.0) / 4000.0;
tflInputTensor->data.f[samplesRead * 9 + 5] = (gZ + 2000.0) / 4000.0;
tflInputTensor->data.f[samplesRead * 9 + 6] = (mX + 400.0) / 800.0;
tflInputTensor->data.f[samplesRead * 9 + 7] = (mY + 400.0) / 800.0;
tflInputTensor->data.f[samplesRead * 9 + 8] = (mZ + 400.0) / 800.0;
```

Problemas

Durante el desarrollo de esta práctica, se obtuvieron los siguientes inconvenientes, principalmente en el momento de diseñar y entrenar los modelos:

1. En el entrenamiento con la versión Local de Tensor Flow, aprovechando que ya se tenian datos consistentes para su uso, el formato CBOR que es un formato para compartir información basado en JSON, pero que para el uso normal en este notebook, se debió transformar en un archivo pandas, este fue un problema leve, al no conocer cómo usar el formato, pero se pudo solucionar instalando una librería que hace uso de este.
2. El uso de la capa Convolucional 1D, a pesar de que es factible en la version completa de TensorFlow, no lo es asi para su version lite enfocada a microcontroladores, por lo que se opta por usar una configuracion similar con convolucionales 2D y una capa extra al inicio para un Reshape de los datos que ingresan, el caso es que esta capa no se ejecuta correctamente en el arduino, genera un error de invoacion de la inferencia al no reconocer un formato de archivo INT32, por lo que se siguieron os pasos presentados en la pagina web [TinyML: Using Arduino Nano 33 BLE sense to run tflite model on Ubuntu 16.04 virtual machine (continuously updated)], que indica cambiar el archivo ubicado en '/home/USER/…/Arduino/libraries/Arduino_TensorFlowLite/src/tensorflow/lite/micro/kernels/strided_slices.cpp', ya que esa capa Reshape es vinculada a este archivo, se modificaron algunas lineas con sus siguientes resultados

```
/*---------添加的代码---------*/
case kTfLiteInt32:
  reference_ops::StridedSlice(op_params,
                              tflite::micro::GetTensorShape(input),
                              tflite::micro::GetTensorData<int8_t>(input),
                              tflite::micro::GetTensorShape(output),
                              tflite::micro::GetTensorData<int8_t>(output));
  break;
/*--------------------------*/
```

En el serial monitor se obtienen los siguientes resultados, aun así se puede ver que trata de identificar el movimiento realizado, pero con errores en los valores, este error no sabemos como solucionarlo.

```
Accelerometer sample rate = 119.00 Hz
Gyroscope sample rate = 119.00 Hz
Magnetic Field sample rate = 20.00 Hz

md_avanzar: ovf
md_derecha: -0.000000
md_izquierda: ovf
md_retroceder: -0.000000

md_avanzar: -0.016501
md_derecha: -0.000000
md_izquierda: ovf
md_retroceder: -0.000000

md_avanzar: ovf
md_derecha: -0.000000
md_izquierda: ovf
md_retroceder: -0.000000

md_avanzar: 0.000000
md_derecha: -0.000000
md_izquierda: ovf
md_retroceder: -0.000000
```

Referencias

Este trabajo se basó en el desarrollo realizado por Ángel Villanueva, en su blog Escornabot: Proyecto para niños con discapacidad visual o ceguera, donde hace uso del acelerómetro del arduino, y a partir de aquí realiza comandos de decisión basado en inclinación.

Tabla 2. Relación de los ángulos de los ejes X e Y.

Instrucción	Inclinación Y	Inclinación X
avanzar / si	(y >= 20º)	(x >= -10º), (x <= 10º)
retroceder / no	(y <= -20º)	(x >= -10º) , (x <= 10º)
izquierda	(y >= -10º) , (y <= 10º)	(x >= 15º)
derecha	(y >= -10º) , (y <= 10º)	(x <= -15º)

centro	(y >= -10º) , (y <= 10º)	(x >= -10º), (x <= 10º)

La mejora que proponemos, es que estos movimientos se puedan identificar con un uso más completo de varios sensores, y con la posibilidad de ampliar a futuro los gestos realizados, sin limitar sus opciones, mediante datos que están basados en series de tiempos e inferidos mediante tecnología de inteligencia artificial con tflite para microcontroladores.