MUÑECA

```
/** ELCO Project -- uVision
* Code for the ESP32 that will go in the wrist
* Features:
* - BLE Client
* - Motor Control
* - Mode Selection
*/
#include "BLEDevice.h"
const int pinMotor = 18; // Use GPIO number. See ESP32 board pinouts
int measure; // Variables
int modeSel = 1;
// The remote service we wish to connect to.
static BLEUUID serviceUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b");
// The characteristic of the remote service we are interested in.
static BLEUUID charUUID("beb5483e-36e1-4688-b7f5-ea07361b26a8");
static BLEAddress *pServerAddress;
static boolean doConnect = false;
static boolean connected = false;
static BLERemoteCharacteristic* pRemoteCharacteristic;
/**
* Check for the BLE connection
*/
bool connectToServer(BLEAddress pAddress) {
Serial.print("Forming a connection to ");
Serial.println(pAddress.toString().c_str());
BLEClient* pClient = BLEDevice::createClient();
Serial.println(" - Created client");
// Connect to the remove BLE Server.
pClient->connect(pAddress);
Serial.println(" - Connected to server");
```

```
// Obtain a reference to the service we are after in the remote BLE server.
BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
if (pRemoteService == nullptr) {
Serial.print("Failed to find our service UUID: ");
Serial.println(serviceUUID.toString().c_str());
return false;
}
Serial.println(" - Found our service");
// Obtain a reference to the characteristic in the service of the remote BLE server.
pRemoteCharacteristic = pRemoteService->getCharacteristic(charUUID);
if (pRemoteCharacteristic == nullptr) {
Serial.print("Failed to find our characteristic UUID: ");
Serial.println(charUUID.toString().c_str());
return false;
}
Serial.println(" - Found our characteristic");
} // connectToServer
/**
* Scan for BLE servers and find the first one that advertises the service we are looking for.
*/
class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
void onResult(BLEAdvertisedDevice advertisedDevice) { //Called for each advertising BLE server.
Serial.print("BLE Advertised Device found: ");
Serial.println(advertisedDevice.toString().c_str());
// We have found a device, let us now see if it contains the service we are looking for.
if (advertisedDevice.haveServiceUUID() && advertisedDevice.getServiceUUID().equals(serviceUUID))
{
Serial.print("Found our device! address: ");
advertisedDevice.getScan()->stop();
pServerAddress = new BLEAddress(advertisedDevice.getAddress());
doConnect = true;
```

```
} // Found our server
} // onResult
}; // MyAdvertisedDeviceCallbacks
void scanner(){
// Retrieve a Scanner and set the callback we want to use to be informed when we
// have detected a new device. Specify that we want active scanning and start the
// scan to run for 30 seconds.
BLEScan* pBLEScan = BLEDevice::getScan();
pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
pBLEScan->setActiveScan(true);
pBLEScan->start(10);
} // scanner
/**
* Setup
*/
void setup() {
Serial.begin(115200); // Serial communication
pinMode(pinMotor, OUTPUT); // Motor output
// Create the BLE Device
BLEDevice::init("");
Serial.println("Starting BLE Client application...");
} // End of setup.
/**
* Main loop function
*/
void loop() {
// If the flag "doConnect" is true then we have scanned for and found the desired
// BLE Server with which we wish to connect. Now we connect to it. Once we are
// connected we set the connected flag to be true.
if (doConnect == true) {
if (connectToServer(*pServerAddress)) {
```

```
Serial.println("We are now connected to the BLE Server.");
connected = true;
} else {
Serial.println("We have failed to connect to the server; there is nothin more we will do.");
}
doConnect = false;
} else if (connected == false) {
scanner();
}
//If we are connected to a peer BLE Server, read the characteristic each time we are reached
//with the current time since boot.
if (connected) {
uint8_t rxValue = pRemoteCharacteristic->readUInt8();
measure = rxValue*4;
distance(measure);
}
delay(10); // Delay between loops.
} // End of loop
/**
* Motor control function
*/
void distance(int measure) {
if (measure > 300) {
Serial.print("No hay peligro: ");
Serial.print(measure);
Serial.println(" ");
digitalWrite(pinMotor, LOW);
} else if ((measure <= 300) && (measure > 200)) {
Serial.print("Objeto detectado a: ");
Serial.print(measure);
Serial.println(" cm ");
```

```
digitalWrite(pinMotor, HIGH);
delay(600);
digitalWrite(pinMotor, LOW);
} else if ((measure <= 200) && (measure > 150)) {
Serial.print("Objeto cercano a: ");
Serial.print(measure);
Serial.println(" cm ");
digitalWrite(pinMotor, HIGH);
delay(450);
digitalWrite(pinMotor, LOW);
} else if (measure <= 150 && (measure > 100)) {
Serial.print("Cuidado, que te das en: ");
Serial.print(measure);
Serial.println(" cm ");
digitalWrite(pinMotor, HIGH); // Run in high speed
delay(375);
digitalWrite(pinMotor, LOW);
} else if (measure <= 100 ) {
Serial.print("Parate boludooo que estas a: ");
Serial.print(measure);
Serial.println(" cm ");
digitalWrite(pinMotor, HIGH); // Run in high speed
delay(200);
digitalWrite(pinMotor, LOW);
}
delay(measure);
} // End of distance
```

GAFAS

```
/** ELCO Project -- uVision
* Code for the ESP32 that will go on the glasses
* Features:
* - BLE Server
* - Sensor Read
* - Buzzer Control
*/
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include <BLE2902.h>
#define SERVICE_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
const int pinSensor = 34; // GPIO number. See ESP32 board pinouts
const int pinBuzz = 18; // GPIO number. See ESP32 board pinouts
int pulse1, sensorValue; // Variables
uint8_t txValue = 0; // BL-->TX
BLECharacteristic *pCharacteristic;
// Para antirebotes
/*
const int timeThreshold = 150;
const int intPin = 2;
volatile int ISRCounter = 0;
int counter = 0;
long timeCounter = 0;
bool encendido = false;
*/
void setup () {
//if digital read pin boton = 1 then encendido = not(encendido) ... if encendido funcion distancia else nada
```

```
Serial.begin(115200); //Serial communication
pinMode(pinSensor, INPUT); //Sensor input
pinMode(pinBuzz, OUTPUT); //Buzzer output
/*
//antirebote boton
pinMode(intPin, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(intPin), debounceCount, FALLING);
attachInterrupt(digitalPinToInterrupt(intPin), onoff, FALLING);
*/
// Create the BLE Device
BLEDevice::init("ESP32");
// Create the BLE Server
BLEServer *pServer = BLEDevice::createServer();
// Create the BLE Service
BLEService *pService = pServer->createService(SERVICE_UUID);
// Create a BLE Characteristic
pCharacteristic = pService->createCharacteristic(
CHARACTERISTIC_UUID,
BLECharacteristic::PROPERTY_NOTIFY
);
pCharacteristic->addDescriptor(new BLE2902());
// Start the service
pService->start();
// Start advertising
BLEAdvertising *pAdvertising = pServer->getAdvertising();
pAdvertising->addServiceUUID(pService->getUUID());
pAdvertising->start();
Serial.println("BLE Server defined!");
}
void loop () {
//antirebote
```

```
/*if (counter != ISRCounter)
{
counter = ISRCounter;
//Serial.println(counter);
}
if (encendido = true)
{*/
read_sensor(); // Read the sensor value
txValue = sensorValue / 4; // Value to send (8 bits, max 256)
pCharacteristic->setValue(&txValue,1);
pCharacteristic->notify();
Serial.print("*** Sent Value: ");
Serial.print(txValue);
Serial.println(" ***");
distance(sensorValue);
delay(10); // This delay time changes by 50 for every sensor in the chain. For 5 sensors this will be 250
//}
// else {}
}
// Function to read the ultrasonic sensor value every loop
void read_sensor() {
pulse1 = pulseIn(pinSensor, HIGH); // Measurement for PW input in uS
sensorValue = 2.54 * (pulse1 / 147); // Inch to cm conversion
}
/**
* Funcion encendido/apagado
*/
/*
void onoff()
{
```

```
encendido = not(encendido);
}
/**
* Funcion antirebote boton encendido/apagado
void debounceCount()
{
if (millis() > timeCounter + timeThreshold)
{
ISRCounter++;
timeCounter = millis();
}
}
*/
/**
* Buzzer control function
*/
void distance(int measure) {
if (measure > 300) {
Serial.print("No hay peligro: ");
Serial.print(measure);
Serial.println(" ");
digitalWrite(pinBuzz, LOW);
} else if ((measure <= 300) && (measure > 200)) {
Serial.print("Objeto detectado a: ");
Serial.print(measure);
Serial.println(" cm ");
digitalWrite(pinBuzz, HIGH);
delay(600);
digitalWrite(pinBuzz, LOW);
} else if ((measure <= 200) && (measure > 150)) {
Serial.print("Objeto cercano a: ");
```

```
Serial.print(measure);
Serial.println(" cm ");
digitalWrite(pinBuzz, HIGH);
delay(450);
digitalWrite(pinBuzz, LOW);
} else if (measure <= 150 && (measure > 100)) {
Serial.print("Cuidado, que te das en: ");
Serial.print(measure);
Serial.println(" cm ");
digitalWrite(pinBuzz, HIGH); // Run in high speed
delay(375);
digitalWrite(pinBuzz, LOW);
} else if (measure <= 100 ) {
Serial.print("Parate boludooo que estas a: ");
Serial.print(measure);
Serial.println(" cm ");
digitalWrite(pinBuzz, HIGH); // Run in high speed
delay(200);
digitalWrite(pinBuzz, LOW);
}
delay(measure);
} // End of distance
```