**Manual: Carrito controlado por celular con ESP32 y Python**

**Ing. Sandra Patricia González**

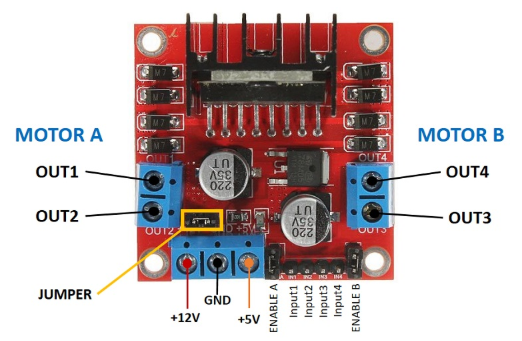
**1. Materiales y explicación de cada componente**

* **ESP32**



Tomado de: <https://images.theengineeringprojects.com/image/webp/2024/03/esp32-pinout.jpg.webp?ssl=1>  
Es una placa de desarrollo con un microcontrolador que incluye **WiFi y Bluetooth integrados**.  
Será el **cerebro del carrito**, encargado de recibir las órdenes enviadas desde el celular y controlar el movimiento de los motores.

* **Puente H L298N**

**Video:** [**https://www.youtube.com/watch?v=bIUswhZ4xBQ&ab\_channel=AprendiendoconGustavo**](https://www.youtube.com/watch?v=bIUswhZ4xBQ&ab_channel=AprendiendoconGustavo)  


Es un módulo electrónico que permite **controlar la dirección y velocidad** de los motores de corriente continua.  
Cada L298N puede controlar **dos motores**, por eso se necesitan dos para manejar los cuatro motores del carrito.

* **Motorreductores de 6V con llantas (x2)**



**Video:** [**https://www.youtube.com/watch?v=PuFb6B7g\_PU&ab\_channel=Profe.Alencaste**](https://www.youtube.com/watch?v=PuFb6B7g_PU&ab_channel=Profe.Alencaste)

**Video:** [**https://www.youtube.com/watch?v=sFv9Su5vvm8&ab\_channel=FARADAYPERU**](https://www.youtube.com/watch?v=sFv9Su5vvm8&ab_channel=FARADAYPERU)  
Son pequeños motores de corriente continua con una caja reductora que disminuye la velocidad pero aumenta la fuerza (**torque**).  
Los reductores permiten que el carrito tenga la potencia suficiente para moverse.

* **Baterías de litio 3.7V 1800mAh (x2)**



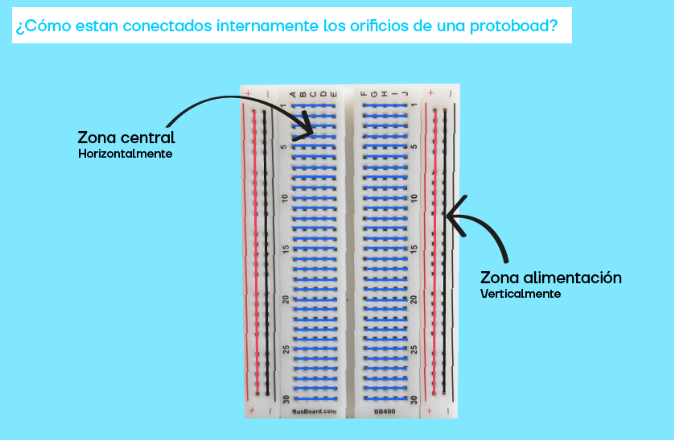
Son la fuente de energía del proyecto.

* + Al conectarlas en **serie**, aumenta el voltaje.
  + Al conectarlas en **paralelo**, aumenta la duración.
  + Con ellas se alimentan los puentes H y el ESP32.
* **Porta baterías (x1)**
*   
  Es el soporte donde se colocan las baterías para conectarlas de forma segura y ordenada.
* **Cables Dupont**



Son cables de conexión rápida que permiten unir la ESP32, los módulos L298N y los demás componentes.  
Se utilizan para hacer las conexiones eléctricas sin necesidad de soldar.

* **Protoboard (opcional)**

  
Placa de pruebas que permite realizar conexiones temporales sin necesidad de soldadura. Facilita el montaje y pruebas previas.

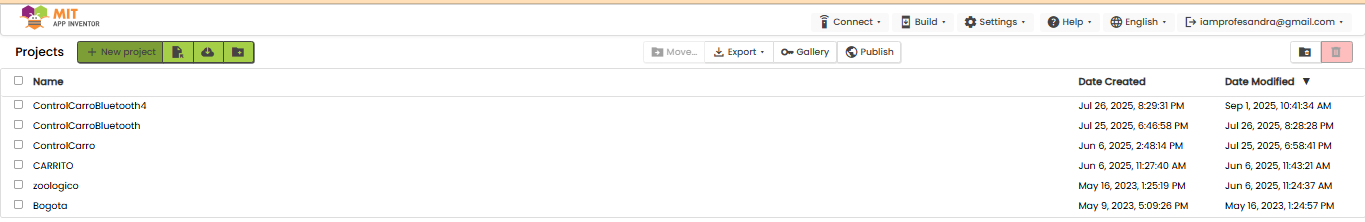
* **Celular Android con MIT App Inventor**



El celular será el **control remoto del carrito**.  
En MIT App Inventor se diseña una aplicación con botones que enviarán comandos al ESP32 por Bluetooth.

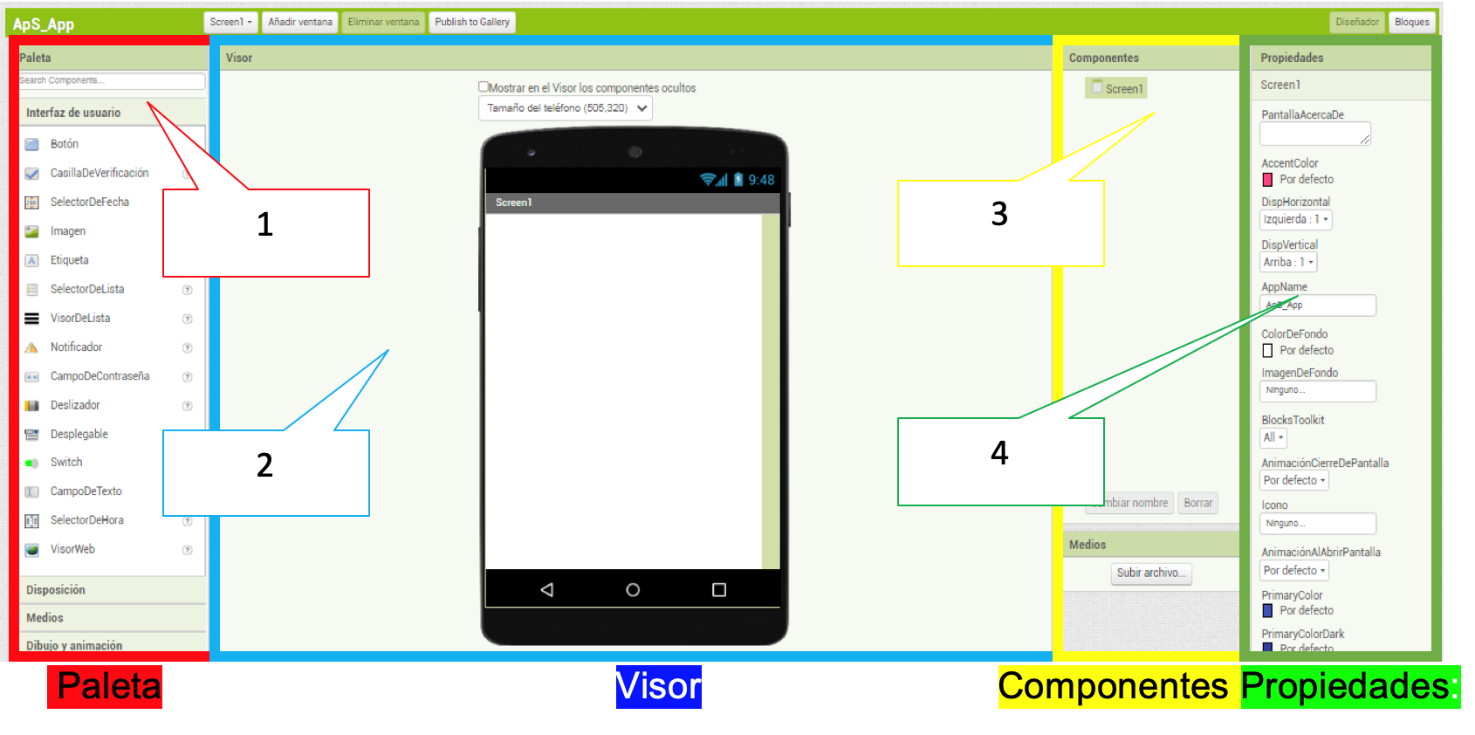
Pasos:

1. Ingresa a la página <https://appinventor.mit.edu/>
2. Crear una cuenta.
3. Se abre una ventana y debes cliquear en New Project:

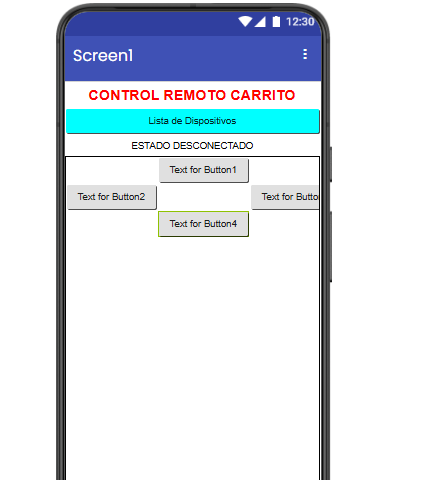


Dale un nombre al proyecto por ejemplo ControlCarritoDosLlantas.

1. Te va a aparecer una ventana como la siguiente:



1. **PALETA:**Agrupados en 12 categorías, ofrece multitud de elementos para integrarlos en nuestra aplicación arrastrándolos a la ventana Visor.
2. **VISOR**. Muestra una pantalla similar a un dispositivo móvil para ir alojando los elementos de la paleta y comprobar su disposición.
3. **COMPONENTES**.  Aparecen los elementos seleccionados en el visor en forma de árbol. Incluye una ventana inferior “Medios” para subir archivos multimedia.
4. **PROPIEDADES**. La cuarta columna permite la configuración de los componentes seleccionados anteriormente: color y alineación de texto, visible, etc. bien para la aplicación en general o bien para cada una de las pantallas de **nuestra** app.
5. En la sección **PALETA🡪User interface**, selecciona **Label** y arrástrala a la ventana **VISOR.** y en la ventana **PROPIEDADES:**
   1. **Width** selecciona **Fill parent**, esto hará que la etiqueta se vea en todo la pantalla.
   2. **Text** escribe CONTROL DE CARRITO.
   3. **Font Size**: 20
   4. **FontBold** Chequeado
   5. **Text Color**: Rojo
   6. **TextAlignment**: Center:1
6. En la sección **PALETA**, selecciona **ListPicker** y arrástrala a la ventana **VISOR.** y en la ventana **PROPIEDADES:**
   1. **Width** selecciona **Fill parent**, esto hará que la etiqueta se vea en todo la pantalla.
   2. **Text** escribe Lista de Dispositivos (al desplegar éste botón aparece la listas de dispositivos que se han emparejado previamente mediante bluethooth)
   3. **BackgroundColor** Azul
7. En la sección **PALETA**, selecciona **Label** y arrástrala a la ventana **VISOR.** y en la ventana **PROPIEDADES:**
   1. **Width** selecciona **Fill parent**, esto hará que la etiqueta se vea en todo la pantalla.
   2. **Text** escribe ESTADO DESCONECTADO.
   3. **Font Size**: 20
   4. **FontBold** Chequeado
   5. **Text Color**: Rojo
   6. **TextAlignment**: Center:1
8. En la sección **PALETA 🡪sección Layout**, selecciona **TableArrangement1** y arrástrala a la ventana **VISOR.** y en la ventana **PROPIEDADES:**
   1. Columns: 3
   2. Rows: 3
   3. Height: Fill parent
   4. Width: Fill parent
9. En la sección **PALETA 🡪User Interface**, selecciona **Button** y arrastra 4 botones a la ventana **VISOR** como lo muestra la imagen**:**

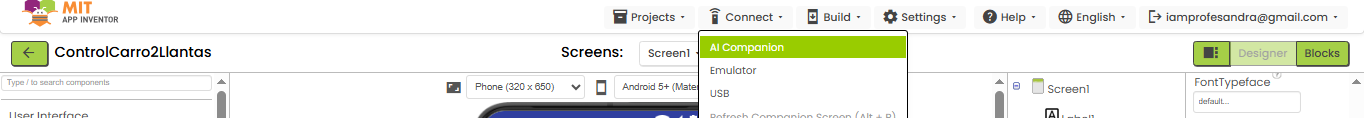


1. Por cada botón haz lo siguiente:

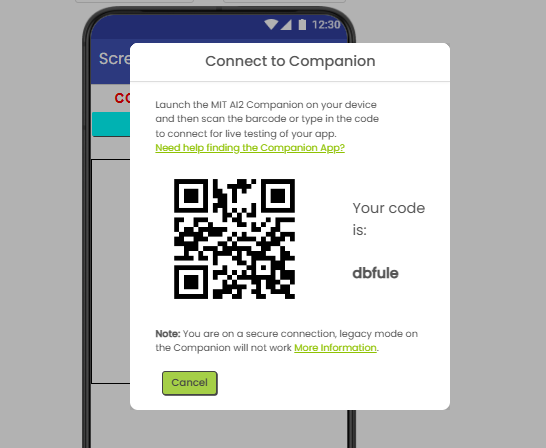
En la ventana **PROPIEDADES:**

* 1. **Text** copia y pega por cada botón las siguientes imágenes 🔼, 🔽, ◀️, ▶️
  2. **Font Size**: 60
  3. **Shape**: rounded

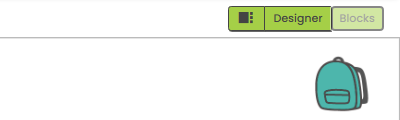
1. En la sección **PALETA 🡪Connectivity** selecciona BluetoothCliente y arrástralo dentro de la ventana, no debe quedar visible.
2. En la sección **PALETA 🡪Sensor** selecciona OrientationSensor y arrástralo dentro de la ventana, no debe quedar visible.
3. Para ver vista previa debes:
4. Descargar la App MIT AI2 Companion.
5. En el pc selecciona Connect 🡪AI Companion



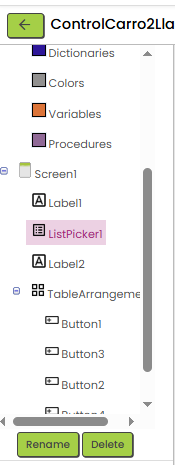
1. Debe aparecer algo como esto:



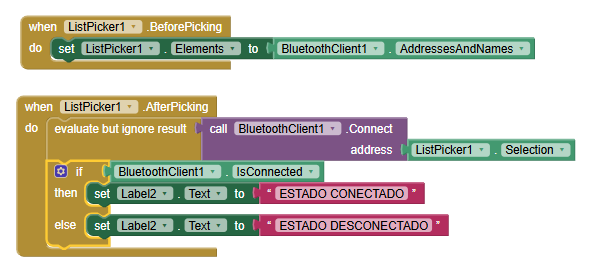
1. Escanea el código QR con la aplicación de App Inventor.
2. Ya debes poder ver en tu celular lo que estás viendo en tu pantalla, los cambios que realices se verán reflejados en tiempo real en tu celular.
3. Programación: Haz clic en la esquina superior derecha sobre el botón Blocks, como se muestra en la siguiente figura:



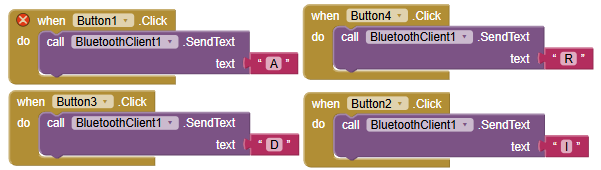
Busca en el panel Izquierdo ListPicker1



Programa los siguientes módulos:



Programación de los Botones:



**2. Alistamiento de la ESP32:**

**🔹 2.1. Instalar MicroPython en el ESP32**

Antes de cualquier librería, tu ESP32 debe tener **MicroPython** cargado (no viene de fábrica).

1. Conecta tu ESP32 al computador por USB.
2. Abre **Thonny IDE**.
3. Ve a: **Herramientas → Opciones → Intérprete**.
4. En “Ejecutar en”, selecciona **ESP32** y el puerto COM que corresponda, para saber el puerto COM debes revisar el Panel de Control de Windows**→** Administrador de dispositivos.
5. Si tu ESP32 no tiene firmware de MicroPython, haz clic en **Instalar firmware o actualizar firmware**.
6. Descarga la versión más reciente de **MicroPython para ESP32** desde:  
   👉 https://micropython.org/download/esp32/
7. Flashea el firmware.

Listo: ahora tu ESP32 ya entiende Python. 🎉

**🔹 2.2. Librerías necesarias para tu código**

Tu código usa estas librerías:

* machine → viene incluida en MicroPython (no hace falta instalarla).
* bluetooth → también incluida en MicroPython para ESP32.
* ble\_simple\_peripheral → **esta no viene incluida**, es un archivo que debemos agregar manualmente.

**🔹2.3. Instalar ble\_simple\_peripheral.py**

Este archivo es un **script de ejemplo** que implementa un periférico Bluetooth Low Energy (BLE).

**Cómo instalarlo en Thonny:**

1. Abre **Thonny**.
2. Menú → **Archivo → Nuevo archivo**.
3. Copia el contenido del ejemplo oficial de MicroPython para ble\_simple\_peripheral.py

# Este ejemplo demuestra un periférico UART (BLE).

import bluetooth

import random

import struct

import time

from ble\_advertising import advertising\_payload

from micropython import const

# Eventos BLE

\_IRQ\_CENTRAL\_CONNECT = const(1)

\_IRQ\_CENTRAL\_DISCONNECT = const(2)

\_IRQ\_GATTS\_WRITE = const(3)

# Flags de permisos

\_FLAG\_READ = const(0x0002)

\_FLAG\_WRITE\_NO\_RESPONSE = const(0x0004)

\_FLAG\_WRITE = const(0x0008)

\_FLAG\_NOTIFY = const(0x0010)

# Definición de servicio y características UART

\_UART\_UUID = bluetooth.UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")

\_UART\_TX = (

bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),

\_FLAG\_READ | \_FLAG\_NOTIFY,

)

\_UART\_RX = (

bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),

\_FLAG\_WRITE | \_FLAG\_WRITE\_NO\_RESPONSE,

)

\_UART\_SERVICE = (

\_UART\_UUID,

(\_UART\_TX, \_UART\_RX),

)

class BLESimplePeripheral:

def \_\_init\_\_(self, ble, name="mpy-uart"):

self.\_ble = ble

self.\_ble.active(True)

self.\_ble.irq(self.\_irq)

# Registrar servicio GATT

((self.\_handle\_tx, self.\_handle\_rx),) = self.\_ble.gatts\_register\_services((\_UART\_SERVICE,))

self.\_connections = set()

self.\_write\_callback = None

self.\_payload = advertising\_payload(name=name, services=[\_UART\_UUID])

self.\_advertise()

def \_irq(self, event, data):

if event == \_IRQ\_CENTRAL\_CONNECT:

conn\_handle, \_, \_ = data

print("Nueva conexión", conn\_handle)

self.\_connections.add(conn\_handle)

elif event == \_IRQ\_CENTRAL\_DISCONNECT:

conn\_handle, \_, \_ = data

print("Desconectado", conn\_handle)

self.\_connections.remove(conn\_handle)

self.\_advertise()

elif event == \_IRQ\_GATTS\_WRITE:

conn\_handle, value\_handle = data

value = self.\_ble.gatts\_read(value\_handle)

if value\_handle == self.\_handle\_rx and self.\_write\_callback:

self.\_write\_callback(value)

def send(self, data):

for conn\_handle in self.\_connections:

self.\_ble.gatts\_notify(conn\_handle, self.\_handle\_tx, data)

def is\_connected(self):

return len(self.\_connections) > 0

def \_advertise(self, interval\_us=500000):

print("Publicidad BLE iniciada")

self.\_ble.gap\_advertise(interval\_us, adv\_data=self.\_payload)

def on\_write(self, callback):

self.\_write\_callback = callback

def demo():

ble = bluetooth.BLE()

p = BLESimplePeripheral(ble)

def on\_rx(v):

print("RX", v)

p.on\_write(on\_rx)

i = 0

while True:

if p.is\_connected():

for \_ in range(3):

data = str(i) + "\_"

p.send(data)

i += 1

time.sleep\_ms(100)

1. Guarda el archivo con el nombre exacto:
2. ble\_simple\_peripheral.py
3. Y muy importante: **guárdalo en el ESP32**, no en tu PC.
   * En Thonny, al guardar, selecciona **Dispositivo / ESP32**.

Así, cuando ejecutes tu código principal, la línea

from ble\_simple\_peripheral import BLESimplePeripheral

funcionará porque el archivo está en la memoria del ESP32.

**🔹 2.4. Verificación rápida**

Para comprobar que todo quedó bien:

1. En Thonny escribe:

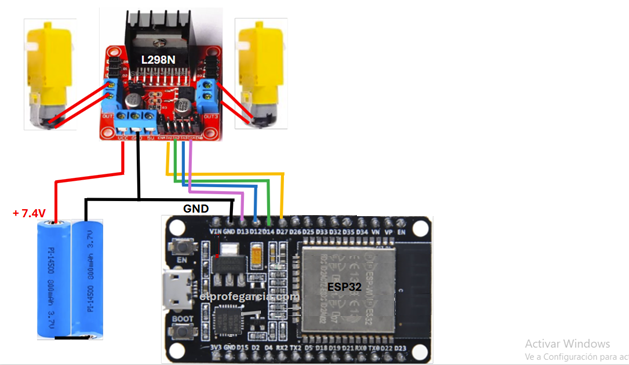
import bluetooth

from ble\_simple\_peripheral import BLESimplePeripheral

print("Bluetooth listo")

1. Ejecuta en el ESP32.
2. Si no da error, tu entorno está correcto.

**. Esquema de conexión eléctrico**

****

* Cada **L298N** controla dos motores. cada L298N tiene 4 pines de control (IN1, IN2, IN3, IN4), que permiten manejar dos motores.  
  Como el carrito tiene dos módulos L298N, necesitamos 8 pines de control en el ESP32.
* **Según el esquema:**

Primer L298N (Motores delanteros, por ejemplo)

IN1 → D27

IN2 → D14

IN3 → D12

IN4 → D13

* **Eso significa que tenemos:**

Motor A → IN1 (D27), IN2 (D14)

Motor B → IN3 (D12), IN4 (D13)

* **🔧 Código:**

from machine import Pin, PWM

import time

import bluetooth

from ble\_simple\_peripheral import BLESimplePeripheral

# Configurar pines motor A con tus pines

in1 = Pin(27, Pin.OUT) # D27

in2 = Pin(14, Pin.OUT) # D14

ena = PWM(Pin(21), freq=1000, duty=0) # D21

# Configurar pines motor B

in3 = Pin(12, Pin.OUT) # D12

in4 = Pin(13, Pin.OUT) # D13

enb = PWM(Pin(22), freq=1000, duty=0) # D22

# Configurar Bluetooth

ble = bluetooth.BLE()

sp = BLESimplePeripheral(ble, "CarritoRobot")

def stop():

ena.duty(0)

enb.duty(0)

def move\_forward(speed):

in1.on()

in2.off()

in3.on()

in4.off()

ena.duty(speed)

enb.duty(speed)

def move\_backward(speed):

in1.off()

in2.on()

in3.off()

in4.on()

ena.duty(speed)

enb.duty(speed)

def turn\_left(speed):

in1.off()

in2.on()

in3.on()

in4.off()

ena.duty(speed)

enb.duty(speed)

def turn\_right(speed):

in1.on()

in2.off()

in3.off()

in4.on()

ena.duty(speed)

enb.duty(speed)

def on\_rx(data):

command = data.decode('utf-8').strip()

speed = 800 # Valor PWM (0-1023)

if command == "A":

move\_forward(speed)

elif command == "R":

move\_backward(speed)

elif command == "I":

turn\_left(speed)

elif command == "D":

turn\_right(speed)

elif command == "S":

stop()

while True:

if sp.is\_connected():

sp.on\_write(on\_rx)

time.sleep\_ms(100)

* **✅ Observaciones:**