

CONECTANDO ARDUINO Y UNITY A TRAVES DEL PUERTO SERIE

Es claro que el primer paso para trabajar con unos datos es obtenerlos y para ello necesitamos un elemento que nos permita leer esos datos que nuestra botonera genera.

Un microcontrolador es básicamente un circuito integrado programable que implementa las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida, con la diferencia que están diseñados para desempeñar una única tarea específica.

En nuestro caso nos hemos decantado por la plataforma Arduino. Digo plataforma porque Arduino no es un microcontrolador como tal si un grupo de placas de desarrollo basadas en microcontroladores la casa Atmel.

Queda claro que los puertos serie son la forma principal de comunicar una placa Arduino con un ordenador. **¿Pero qué son?**

Partimos de la base que se define como **puerto** cualquiera de los interfaces, físicos o virtuales, que permiten la comunicación entre dos ordenadores o dispositivos y que el apellido **serie** viene a decir que la información se transmite como una secuencia de bits “en fila india”.

Un ordenador convencional dispone de varios puertos serie. Los más conocidos son el popular USB (universal serial port) y el ya casi olvidado RS-232. Sin embargo, existen otros muchos, como por ejemplo el RS-485, I2C, SPI, Serial Ata, Pcie Express, Ethernet o FireWire, entre otros.

La mayoría de modelos de Arduino (hay excepciones) no cuentan con un interfaz USB integrado. **¿Pero cómo puede ser que si cuentan con un puerto USB físico entonces?**



Esto se debe a que utilizan **hardware adicional** que les permite **convertir el puerto serie o UART** con el que si cuentan a USB. Los chips más comúnmente utilizados para esta tarea y por lo tanto podéis encontrar en vuestros Arduinos son el Atmega16U2, el FT232RL y el CH340G.

Llegados a este punto queda claro que o bien porque cuentan con una interfaz USB integrada o porque cuentan con un chip que les permite convertir su puerto serie en USB, Arduino cuenta con la capacidad de comunicarse con nuestro PC o cualquier elemento que actúe como anfitrión.

¿Y en el otro lado que?

Cuando conectamos Arduino a un ordenador mediante un cable USB, el ordenador instala un puerto serie virtual (**COM**) desde el cual podemos acceder al puerto serie de Arduino. Por este motivo es necesario instalar en el ordenador los drivers del chip convertidor Serie-USB con el que cuente nuestra placa, aunque si el chip es el Atmega16U2 o el FT232RL la instalación de los drivers del mismo se realizara junto con la instalación del propio IDE de Arduino.

CONFIGURACIÓN DE UNITY (ANTES DE EMPEZAR)

Nuestra principal herramienta en Unity, plataforma de desarrollo con la que vamos a trabajar, a la hora de hacer uso de ese puerto serie virtual, será la clase **SerialPort** incluida en namespace **System.IO.Ports**, que contiene todas las clases para controlar los puertos serie. En nuestro caso esa clase será la encargada de mediar en dicha comunicación entre lo que llega por el puerto serie y nuestro entorno.

Antes de nada debemos asegurarnos que el Nivel de Compatibilidad de la API que tenemos seleccionado en las opciones de configuración de Unity es **la .NET 2.0** y no **la .NET 2.0 subset**. Esto se debe básicamente a que la versión subset es algo así como una versión reducida que no cuenta con todas las funciones del framework **de Microsoft .Net** pero cuenta con la ventaja de ser más liviana. Por lo que el sentido común dice que si puede vivir sin el conjunto de funciones completo es recomendable usar el nivel subset. En nuestro caso en cambio necesitamos contar con esas funcionalidades completas para poder hacer uso del puerto serie.

Una vez hecho este cambio vamos a estudiar por separado el código a implementar en Arduino y su homónimo en Unity.

ARDUINO

Como ya hemos comentado la mayoría de los modelos de placas Arduino disponen de un conector USB o Micro USB conectado a uno de sus puertos serie (UART); lo que simplifica el proceso de conexión con un ordenador.

Para manejar el puerto serie en Arduino, debemos leer a fondo la referencia de Arduino: <http://arduino.cc/en/Reference/Serial> que son las funciones que tenemos disponibles para trabajar con el puerto serie.

En nuestro caso se ha optado por escribir todo en una misma línea utilizando el método `Serial.print()`, **separando los datos a mandar mediante “comas”**. De esta manera Unity podrá **leer una línea entera y almacenarla** en un String. Más tarde podremos dividir ese String tomando como referencia las “comas” añadidas y almacenar los datos obtenidos en un Array que nos ayude a tenerlos accesibles de manera independiente.

Los funciones del puerto serie que vamos a utilizar son las siguientes:

begin() à Establece la velocidad de datos en bits por segundo (baudios) para la transmisión de datos en serie.

print() à Imprime datos al puerto serie como texto ASCII, también permite imprimir en otros formatos.

println() à Imprime datos en el puerto serie como texto ASCII legible por humanos seguido de un carácter de retorno de carro (ASCII 13, o ‘\ r’) y un carácter de nueva línea (ASCII 10 o ‘\ n’).

flush() à Espera hasta la transmisión completa de los datos salientes.

De esta manera tendríamos algo como lo siguiente (pasando por alto la lectura de los datos claro):

```
Sketch_1111144.g
void setup() {
  Serial.begin(9600);
}

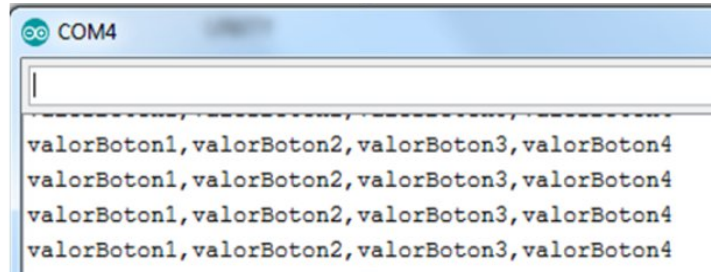
void loop() {

  Serial.flush();

  Serial.print(valorBoton1);
  Serial.print(",");
  Serial.print(valorBoton2);
  Serial.print(",");
  Serial.print(valorBoton3);
  Serial.print(",");
  Serial.println(valorBoton4);

  delay(50);
}
```

Si con este sketch abrimos la ventana del puerto serie en nuestro IDE obtendríamos lo siguiente.



```
COM4  
valorBoton1, valorBoton2, valorBoton3, valorBoton4  
valorBoton1, valorBoton2, valorBoton3, valorBoton4  
valorBoton1, valorBoton2, valorBoton3, valorBoton4  
valorBoton1, valorBoton2, valorBoton3, valorBoton4
```

En el apartado de esta entrada dedicado a la configuración de Unity ya hemos comentado que el namespace **System.IO.Ports** contiene todas las clases para controlar los puertos serie y por ello lo primero que debemos es incluirla al inicio de nuestro Script.

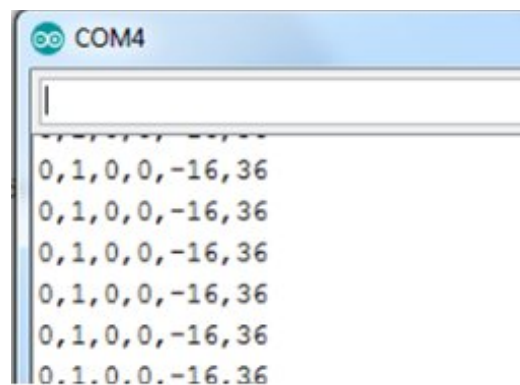
En este caso vamos a poner como **ejemplo** una **botonera** que cuenta con:

- **4 pulsadores** que son leídos como 0 o 1 por el arduino
- **1 joystick** que genera dos valores analógicos (entre -5 y 5 v).



En la imagen inferior se puede apreciar el montaje realizado utilizando un joystick externo a la botonera. En este montaje se ha utilizado un **Arduino nano**. Como apunte recordar que en Arduino podemos aplicar una resistencia pull-up en las entradas digitales a través de código lo que nos permitiría eliminarlas físicamente de nuestro montaje.

Como hemos visto los valores leídos por nuestro Arduino estarán escritos en el puerto serie sin realizar saltos de línea y separados por comas. Si realizamos abrimos el puerto serie en el IDE de arduino, podremos ver algo como lo siguiente:



```
COM4
0,1,0,0,-16,36
0,1,0,0,-16,36
0,1,0,0,-16,36
0,1,0,0,-16,36
0,1,0,0,-16,36
0.1.0.0.-16.36
```

Si analizamos cualquiera de las líneas generadas podemos comprobar, que en el instante de la captura, el segundo botón se encuentra pulsado (leemos un 1). Los dos últimos valores corresponden al valor leído por el **ADC** del Arduino en sus respectivas entradas. El conversor analógico digital de Arduino tiene una resolución de **10 bits**, por lo que nos devolverá **enteros entre 0 y 1023**. Si queremos generar otro rango de valores únicamente deberemos realizar un mapeado o aplicar una simple regla de tres.

En Unity el código quedaría de la siguiente manera:

```
using UnityEngine;
using System;
using System.IO.Ports; //incluimos el namespace Sustum.IO.Ports

public class MandoFull: MonoBehaviour {

    SerialPort serialPort = new SerialPort("COM4", 9600); //Inicializamos el puerto serie

    void Start () {

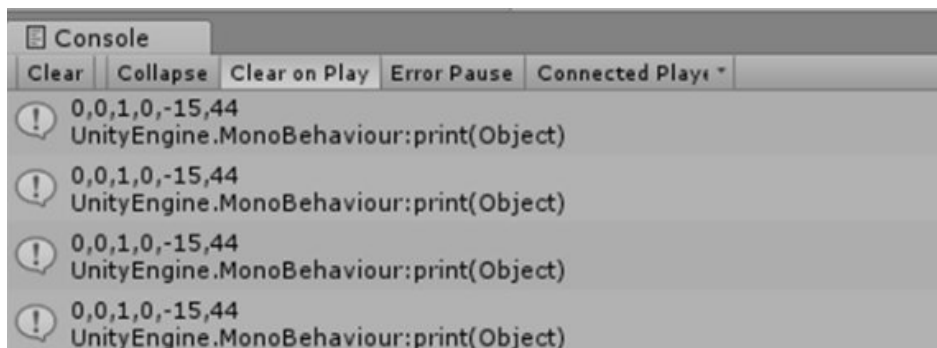
        serialPort.Open(); //Abrimos una nueva conexión de puerto serie
        serialPort.ReadTimeout = 1; //Establecemos el tiempo de espera cuando una operación de
lectura no finaliza
    }
}
```

```

void Update () {
    if (serialPort.IsOpen) //comprobamos que el puerto esta abierto
    {
        try //utilizamos el bloque try/catch para detectar una posible excepción.
        {
            string value = serialPort.ReadLine(); //leemos una linea del puerto serie y
            print(value); //printeamos la linea leida para verificar que leemos el dato
            string[] vec6 = value.Split(','); //Separamos el String leido valiendonos
            //de las comas y almacenamos los valores en un
            array.
        }
        catch
        {
        }
    }
}

```

Una vez hecho esto podremos utilizar la consola de Unity para verificar que el valor leído y “printeado” es correcto. Si todo va bien deberemos ver algo como lo siguiente:



Como se puede observar obtenemos los mismos datos que ya veíamos en la ventana del puerto serie con la que cuenta el IDE de Arduino.

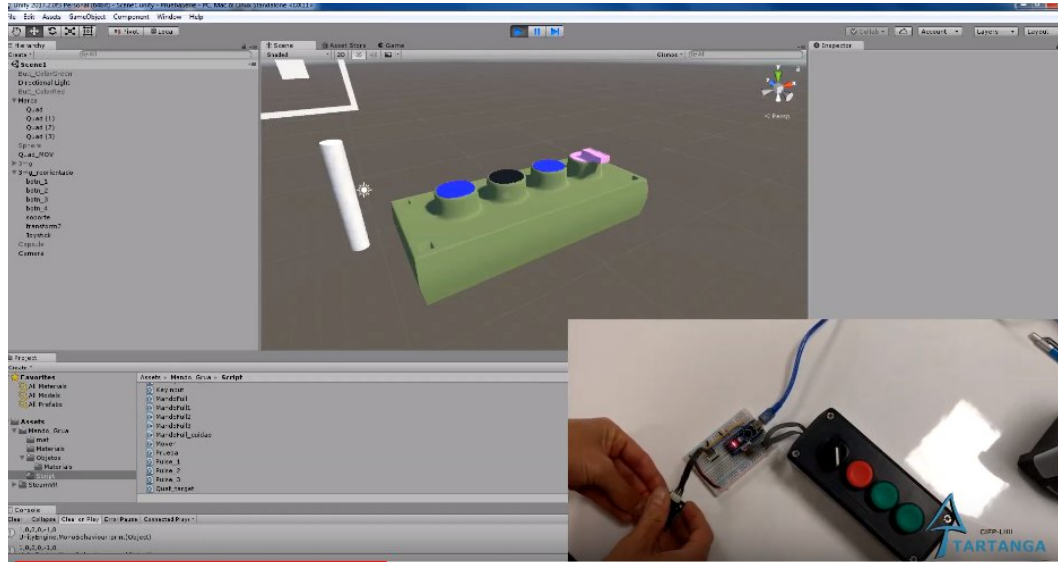
DEL MANDO FÍSICO AL MANDO VIRTUAL

¿Y ahora que tengo los datos en Unity que? En nuestro caso y siguiendo con el objetivo principal del proyecto, hemos modelado el mando utilizado en el montaje descrito anteriormente y los hemos introducido en Unity. De esta manera podremos utilizar los datos que recibimos para animar el mando virtual y que actué como espejo del pequeño mando real del montaje realizado.

Como una imagen vale más que mil palabras, os dejamos un vídeo que da muestra de lo que se ha descrito en esta entrada. Os dejamos también el código tanto de la parte de

Arduino como del Script de Unity con el objetivo de facilitar el trabajo de quien lo requiera.

RESULTADO



Link: https://www.youtube.com/watch?v=4QsFm3H7PQc&feature=emb_title

Codigo Arduino

```
const int pinBoton1= 2; //declaramos constantes con numero de pin para los botones
const int pinBoton2= 3;
const int pinBoton3= 4;
const int pinBoton4= 5;

int valorBoton1= 0; //declaramos variables para almacenar el estado de los botones
int valorBoton2= 0;
int valorBoton3= 0;
int valorBoton4= 0;
int joyX;
int joyY;
int joyXMap;
int joyYMap;

void setup(void){
  Serial.begin(9600);
  pinMode (pinBoton1, INPUT);
  pinMode (pinBoton2, INPUT);
  pinMode (pinBoton3, INPUT);
  pinMode (pinBoton4, INPUT);
}

void loop(void){

  valorBoton1=digitalRead (pinBoton1); //leemos pines donde hemos conectado los botones
  valorBoton2=digitalRead (pinBoton2);
  valorBoton3=digitalRead (pinBoton3);
  valorBoton4=digitalRead (pinBoton4);
  joyX=analogRead (A0); //leemos las entradas analógicas donde hemos conectado el Joystick
```

```
joyY=analogRead (A1);

joyXMap=map(joyX, 0, 1024, -45, 45); //mapeamos los valores de Joustick para darle el rango deseado
joyYMap=map(joyY, 0, 1024, -45, 45);

Serial.flush(); //escribimos los valores en el serial
Serial.print(valorBoton1);
Serial.print(",");
Serial.print(valorBoton2);
Serial.print(",");
Serial.print(valorBoton3);
Serial.print(",");
Serial.print(valorBoton4);
Serial.print(",");
Serial.print(joyXMap);
Serial.print(",");
Serial.print(joyYMap);
Serial.println();

delay(50);
}
```

Codigo Unity

```
public class MandoFull3: MonoBehaviour {

    public GameObject botn_1;
    public GameObject botn_2;
    public GameObject botn_3;
    public GameObject botn_4;
    public GameObject Joystick;
    public float RotY = 90f;

    public float PosIni = 0.5f;

    SerialPort serialPort = new SerialPort("COM4", 9600);

    public float smooth = 20.0F;

    // Use this for initialization
    void Start () {

        serialPort.Open();
        serialPort.ReadTimeout = 1;
    }

    void Update () {

        if (serialPort.IsOpen)
        {
            try
            {

                string value = serialPort.ReadLine();
                print(value);
                string[] vec6 = value.Split(',');
            }
        }
    }
}
```



```
if ((Convert.ToInt32(vec6[0]))== 1)
{
    botn_1.transform.localPosition = new Vector2(botn_1.transform.localPosition.x, 0f);
}
else { botn_1.transform.localPosition = new Vector2(botn_1.transform.localPosition.x, PosIni); }

if ((Convert.ToInt32(vec6[1])) == 1)
{
    botn_2.transform.localPosition = new Vector2(botn_2.transform.localPosition.x, 0f);
}
else { botn_2.transform.localPosition = new Vector2(botn_2.transform.localPosition.x, PosIni); }

if ((Convert.ToInt32(vec6[2])) == 1)
{
    botn_3.transform.localPosition = new Vector2(botn_3.transform.localPosition.x, 0f);
}
else { botn_3.transform.localPosition = new Vector2(botn_3.transform.localPosition.x, PosIni); }

if ((Convert.ToInt32(vec6[3] ))== 1)
{
    botn_4.transform.localRotation = Quaternion.Euler(0, RotY, 0);
}
else { botn_4.transform.localRotation = Quaternion.Euler(0, 0, 0); }

Quaternion target = Quaternion.Euler(Convert.ToInt32(vec6[4]), 0, Convert.ToInt32(vec6[5]));
Joystick.transform.localRotation = Quaternion.Slerp(Joystick.transform.localRotation, target,
Time.deltaTime * smooth);

}
catch
{

}

}

}
}
```