CONECTANDO ARDUINO Y UNITY HACIENDO USO DEL TRACKER DE LAS HTC VIVE

Unos de los principales retos que presentaba nuestro proyecto era la necesidad de que el entorno o plataforma elegida para el desarrollo del mismo fuese capaz de recoger los datos generados por los elementos a "sensorizar". En esta entrada al igual que en la que se abordó el **uso del puerto serie** como método de conexión; se va a utilizar una **botonera** como fuente de datos.

ESTUDIO DEL ELEMENTO

En nuestro caso se trata una botonera de la casa **Schneider electric** con 7 pulsadores simples o una velocidad, 4 pulsadores dobles o dos velocidades y un botón adicional de parada de emergencia. De estos datos deducimos que nuestra botonera genera 16 salidas digitales independientes.

ELECCIÓN DEL MICROCONTROLADOR

Como ya vimos en una anterior entrada, el microcontrolador sera el encargado de **leer los datos** del elemento a sensorizar (en este caso la botonera). Debido a que cada elemento presentara unas características diferentes, el microcontrolador tendrá que estar elegido en base a las mismas.
Cada modelo de Arduino cuenta con una serie de especificaciones que los harán más o menos indicados dependiendo de las exigencias de la tarea a realizar. En nuestro caso las características indispensables son:

- 16 entradas digitales
- Contar con un puerto USB Nativo

¿Qué es esto de USB Nativo?

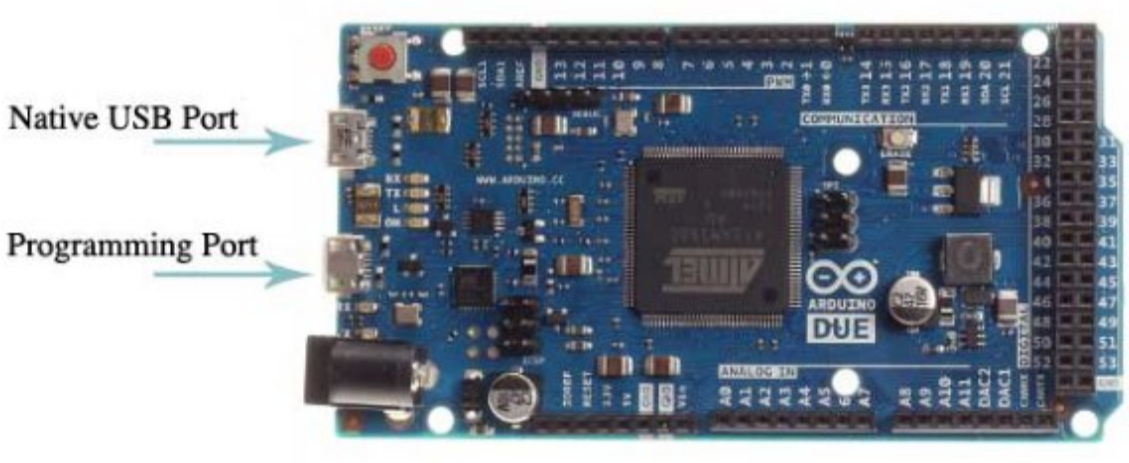
Para empezar necesitamos familiarizarnos con uno de los principios básicos del USB. En las comunicaciones USB contamos con dos elementos: Uno de ellos actua como **"host"** o anfitrión y el otro como **"device"** o dispositivo.
La mayoría de las veces la función de anfitrión la suele llevar a cabo el ordenador y la de dispositivo por los periféricos que conectamos al mismo (impresoras, teclado, webcams,….).

El anfitrión debe saber que dispositivo se va a conectar a él y que características tendrá la comunicación a utilizar. El dispositivo en cambio solo deberá comunicarse con el anfitrión "a su manera". Por lo que siempre será más complejo el trabajo del "host" o anfitrión que el del "device" o dispositivo conectado al mismo. Los Arduinos, como es de suponer, suelen actuar como dispositivos.

En nuestro caso y adelantándonos a que los datos escogidos han de ser enviados a nuestro entorno de desarrollo en el que trabajaremos con los dispotivos de **HTC vive** necesitamos que nuestro microcontrolador pueda actuar como host o anfitrión para el resto de dispositivos.

Para esta labor tendremos que optar por un modelo de Arduino que cuente con un chip más "poderoso" que sea capaz de llevar a cabo ese **roll de anfitrión para otros dispositivos** (en nuestro caso el **Tracker de HTC**).

En el mercado existen varios modelos que cuentan con posibilidad de actuar como anfitriones o "host", entre los que están el arduino Yun, el arduino Zero, el aduino Tian y el Due. Habiéndonos decantado nosotros por este ultimo debido a unas especificaciones más acordes con las necesidades del proyecto. Cabe mencionar que también existen las llamadas **USB host shields** que serán un hardware adicional que nos permite añadir la posibilidad de actuar como anfitriones a modelos que no cuentan con ella.



En cuanto a la programación, tal y como lo hicimos en la entrada que estudiábamos el envió de datos vía puerto serie, vamos a ver por separado el codigo correspondiente a Arduino y el de Scripts de Unity.

ANTES DE EMPEZAR

Antes de meternos a programar nada tenemos que darnos cuenta que ahora contamos con un nueva variable que no conocemos en la ecuacion: **el tracker**.
No podemos programar nada sin comprender el funcionamiento del mismo y hay varias preguntas que debemos responder. ¿que posibilidades nos permite el tracker? ¿Cuantos datos podemos mandar? ¿en que formato?

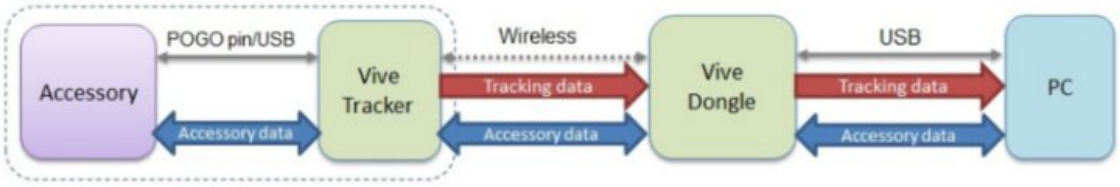
Todos estos datos los podemos encontrar en la guia para desarrolladores del producto que podeis encontrar en el siguiente link:

https://dl.vive.com/Tracker/Guideline/HTC_Vive_Tracker_Developer_Guidelines_v1.3.pdf

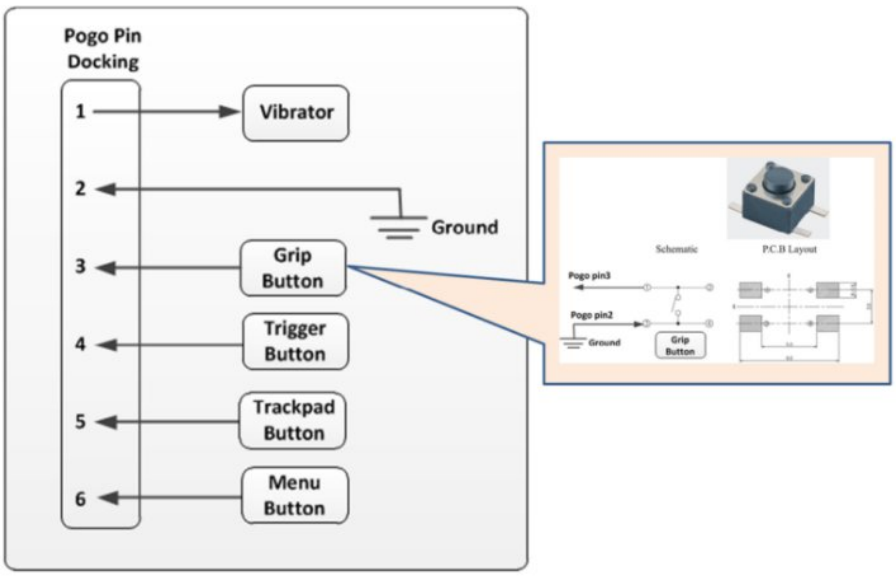
Este documento contiene información sobre cómo usar el HTC Vive Tracker para habilitar la posición seguimiento y transmisión de datos específicos (con o sin el sistema HTC Vive VR). En el tambien se explica que un accesorio adjunto a Vive Tracker (en nuestro cado el Arduino) puede:

- Simular botones del controlador Vive a través los **Pogo pins**.
- **Enviar datos específicos** a una PC a través de la **interfaz USB** de Vive Tracker.

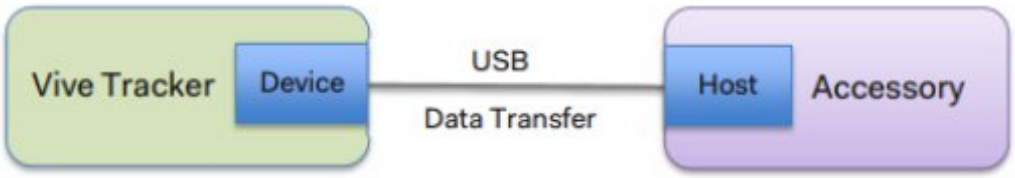
En la documentación descrita en el parrafo anterior se presentan varios modos de uso del Tracker siendo el 5 caso de uso el realmente interesante para nuestro cometido.



En el diagrama anterior se puede ver como podemos utilizar tanto los POGO pins como el USB para "escribir" información en el tracker desde nuestro accesorio (Arduino en nuestro caso) y que este se encargara de mandar via **Dongle** los **datos del accesorio junto los datos de Trackeo** (posicionado) al PC. Las flecha de "Accessory data" hace intuir en la bidireccionalidad de los datos aunque en esta entrada solo abordaremos la transmisión de datos del accesorio al Tracker y no al revés.

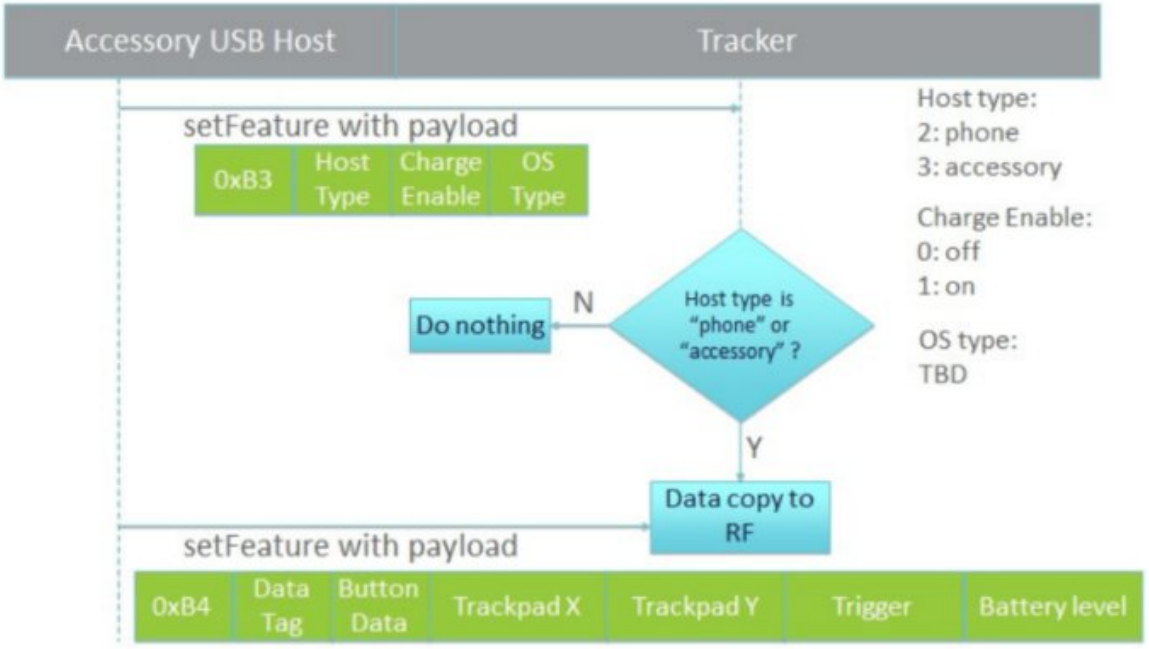


En el caso de hacer uso de los POGO pins no tendremos más que utilizar microruptores para **conectar a tierra los pines que queramos activar**. Los pines cuentan en su interior con resistencias pull-up que nos facilitan mucho el trabajo.
En el caso del USB la cosa se complica un poco más pero no asustarse. En la misma documentación podemos encontrar un apartado dedicado a los "accessory makers" que nos sera de gran ayuda.



En el diagrama se puede apreciar como el Tracker de Vive actuará tendrá el rol de dispositivo en la comunicación USB para transferir datos hacia / desde el accesorio; que deberá actuar como "host" (pero esto no es nuevo para nosotros).

Vale, ahora ya sabemos que nos permite hace el Tracker, **pero ¿cuantos datos podemos mandar y en que formato?**



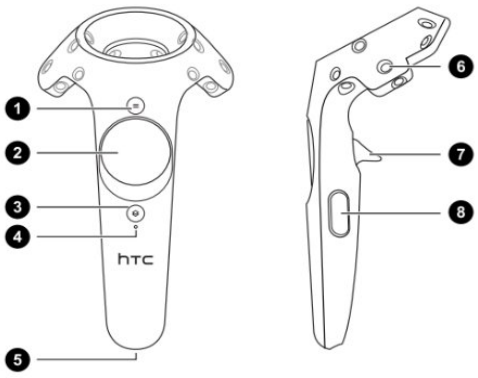
En el diagrama de flujo que podemos en contrar en la guia para desarrolladores se puede ver como contamos con dos comanndos "setfeature" que podriamos traducir como elección de características. El primero (0xB3) lo utilizaremos para completar el byte de tipo Host de manera que el Tracker sepa a que tipo de dispositivo anfitrión a sido conectado. Si en el **byte de Host Type** escribimos un 2 estaremos diciendole que esta

conectado a un telefono y si escribimos un 3 a un accesorio. Los byte de **Charge Enable** y **OS Type** por su parte estaran reservados a requerimiento del propio Tracker.
Una vez de que el Tracker ya sabe a que dispositivo esta conectado podemos continuar con nuestro trabajo de "escritura de datos" en el Tracker. Para ello en la documentación aparece una tabla que nos indica los **bytes** que tenemos **disponibles** para dicha tarea.

SetFeature 0xB4 data format:

Byte Index	Data	Remark
0	Tag Index	Indicates the version of the data you send out. Default value is zero in this version of data format.
1	Button	TRIGGER 0x01 BUMPER 0x02 MENU 0x04 STEAM 0x08 PAD 0x10 PAD_FINGERDOWN 0x20 Reserved 0x40 Reserved 0x80
2	Pad X value	Pad X value, value from -32768 to 32767
3		
4	Pad Y value	Pad Y value, value from -32768 to 32767
5		
6	Trigger Raw	Trigger Raw, value from 0 to 65535
7		
8	Battery Level	Battery Level, Reserved
9		

Viendo la anterior tabla hemos deducido, siempre con la prudencia que exige hacer este tipo de afirmaciones, que HTC no ha desarrollado un nuevo dispositivo de 0 para implementar el Tracker y que simplemente ha "reciclado" los **mandos o controllers** que vienen de serie con el Pack HTC Vive.



1	Menu button
2	Trackpad
3	System button
4	Status light
5	USB charging adapter
6	Tracking sensor
7	Trigger
8	Grip button

Como se puede apreciar los nombres de los botones del mando y los que encontramos dentro del byte Button de nuestro Tracker coinciden (Grip, Menu, pad,…). De la misma manera podemos deducir que los bytes destinados a guardar los datos del gatillo (Trigger) son el 6 y el 7; mientras que los destinado a guardar los valores del Pad (X e Y) son el 2, 3, 4, y 5.

ARDUINO

Antes de empezar dar gracias desde aqui al usuario **matzman666** que ha realizado una modificación a la biblioteca de USBHost tanto para placas Arduino con arquitectura AVR(Uno, Nano, Leonardo) como ARM (due, Zero, Teensy), haciéndola compatible con el Tracker de HTC; facilitándonos el trabajo.

La biblioteca USBHost es la que permite que una placa Arduino, en nuestro caso Due, aparezca como un host USB, permitiéndole comunicarse con periféricos como mouse y teclados USB. Estudiemos entonces el añadido realizado por el usuario matzman666.

Link: https://github.com/matzman666/USBHost-samd

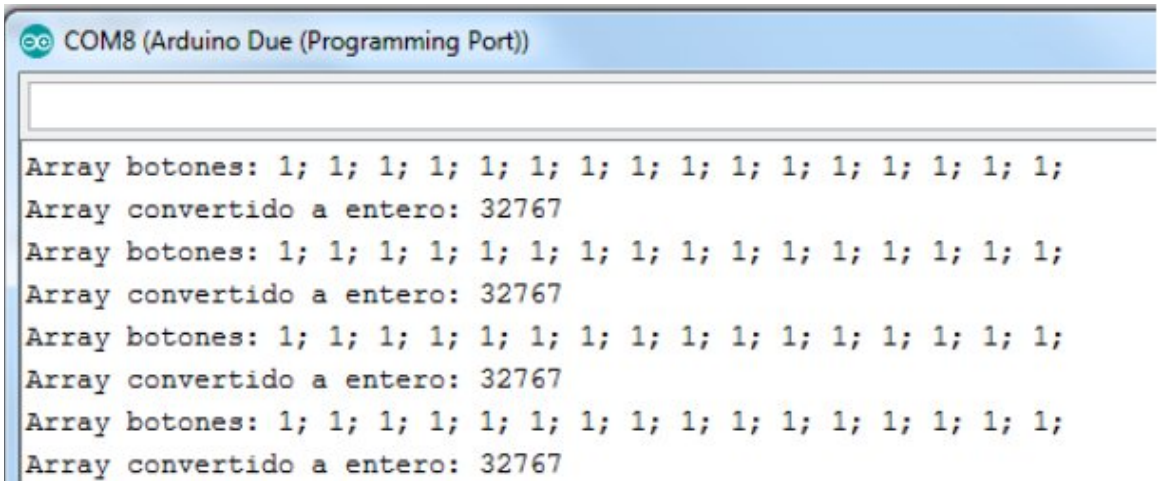
Si abrimos la librería nos encontraremos que el usuario a añadido el archivo **ViveTrackerController.h** dentro de la carpeta src de la libreria. En este archivo se encuentran la clase, las variables y los métodos necesarios para comunicarnos con el Tracker. Si tenéis curiosidad podéis abrir el archivo ViveTrackerController.h en cualquier editor de código y analizarla.
Si no queréis entrar ahí podéis abrir directamente el ejemplo de uso de la librería que podemos encontrar en la carpeta "**Examples"** de la misma, bajo el nombre "**ViveTracker.ino"**.
En el código de ejemplo podemos ver como utilizamos el método **setTrackerStatus** para escribir los valores deseados en el Tracker.

```
// Require vive tracker controller library
#include <ViveTrackerController.h>

// Initialize USB Host Controller
USBHost usb;

// Attach vive tracker controller to USB
ViveTrackerController tracker(usb);


void setup() {
  Serial.begin(115200);
  Serial.println("Program started");
  //delay(200);
}


void loop() {
  tracker.Task(); // Process usb and vive tracker controller tasks
  if (tracker.isConnected()) {
    // Send accessory state to vive tracker.
    uint8_t buttons = VIVETRACKER_BUTTON_GRIP | VIVETRACKER_BUTTON_MENU;
    int16_t padX = 1234;
```

```
    int16_t padY = -333;
    uint8_t trigger = 200;
    uint16_t batteryLevel = 0;
    tracker.setTrackerStatus(buttons, padX, padY, trigger, batteryLevel);
  }
  delay(300);
}
```

Como podemos apreciar la librería nos ha facilitado mucho el trabajo y ahora solo tenemos que preocuparnos de leer los datos de la botonera y darles un formato que nos permita meterlos en los bytes que nos permite el Tracker.

En nuestro caso hemos utilizado los **2 bytes correspondientes al PadX** para enviar nuestros datos. **¿Pero cómo encapsulamos 16 entradas digitales en 2 bytes?** Nos hemos decidido por meter los valores leídos de la botonera en un **array de booleanos,** que luego **convertiremos a entero** mediante un pequeño método que hemos implementado. Si imprimimos el resultado por puerto serie tendríamos algo como lo siguiente.



En la captura podemos ver como el array que obtendremos contendrá los valores de los botones leídos: leemos 1s debido a que estamos utilizando las resistencias pull-up internas de nuestro Arduino por lo que leeremos 0 cuando se encuentren pulsados.

Tenemos que tener en cuenta que en los 2 bytes destinados a almacenar los valores de PadX pueden contener valores de -32768 a 32767. Es decir, que el MSB o bit de mayor de peso esta destinado al signo. Esto supone que para nuestra tarea solo nos son utiles **15 bits** de esos 2 bytes.

Como anteriormente hemos comentado nuestra botonera genera **16 salidas digitales**, por lo que esos 15 bits **no son suficientes** para almacenarlos. La solución que hemos adoptado ha sido enviar ese bit a través del **byte destinado a los botones**.

```
if (tracker.isConnected()) {
  // Send accessory state to vive tracker.
```

```
    uint8_t buttons = arrayValores [15]; //Utilizamos el campo buttons para mandar el bit que no entra en el PadX
    //int16_t padX = 32767; //ya esta rellenado arriba
    int16_t padY = -32768; //rellenamos los campos a enviar (no utilizados)
    uint8_t trigger = 0; //rellenamos los campos a enviar (no utilizados)
    uint16_t batteryLevel = 0; //rellenamos los campos a enviar (no utilizados)

    tracker.setTrackerStatus(buttons, padX, padY, trigger, batteryLevel); //utilizamos el metodo setTrackerStatus de
viveTrackerController.h para "escribir" en el Tracker

  }
  delay(100);
}

  int16_t BoolArrayToInt(bool boolArray[14]) //metodo para convertir el array de booleanos a entero
{
  int result = 0;

  for(int i = 0; i < 15; i++)
  {
    if(boolArray[i])
    {
      result = result | (1 << i);
    }
  }

  return result;
}
```

UNITY

Lo primero a tener en cuenta a la hora de leer nuestro Tracker, es que el Tracker no es el único elemento que envía datos a nuestras estaciones y Dongle, ya que también podremos acceder a los datos enviados por el HMD (Head Mounting Device) así como a los dos mandos incluidos en el Pack.

En la jerarquia SteamVR hay dos tipos de componentes principales. TrackedObjects y TrackedControllers. Los **TrackerControllers** nos servirán para acceder a las **entradas básicas** de nuestros mandos: básicamente leer cuando los **botones del mando** son pulsados. Los **TrackedObjects** por su parte están destinados a ser utilizado en cualquier objeto que sea Trackeado por las estaciones, incluido el HMD y el Tracker. De esa manera hemos usado el TrackedObject para obtener una **representación entera de nuestro controlador** que luego asignamos a **SteamVR_Controller.Device** para obtener algunos métodos específicos del controlador para nuestro uso. Esta forma ademas de permitirnos acceder a otros elementos que no son los controllers o mandos, nos permite tener **más flexibilidad** en términos de lo que realmente podemos obtener de nuestro elemento: leer los valores del Touchpad del mando por ejemplo.

```
using UnityEngine;

public class  LecturaTracker: MonoBehaviour
{
        private SteamVR_TrackedObject trackedObject;


        private void Start()
```

```
        {
                trackedObject = GetComponent<SteamVR_TrackedObject>();
        }

        private void Update()
        {
                var device = SteamVR_Controller.Input((int)trackedObject.index);


        }
}
```

Una vez de que ya tenemos nuestro device asignado, ya podemos hacer uso de los métodos para acceder a los diferentes elementos (botones, pad, trigger,…) de nuestro dispositivo. Si abrimos la clase SteamVR_Controller podremos encontrarlos fácilmente.

```
public bool GetPressDown(ulong buttonMask) { Update(); return (state.ulButtonPressed & buttonMask) != 0 &&
(prevState.ulButtonPressed & buttonMask) == 0; }

public bool GetPressUp(ulong buttonMask) { Update(); return (state.ulButtonPressed & buttonMask) == 0 &&
(prevState.ulButtonPressed & buttonMask) != 0; }

public bool GetPress(EVRButtonId buttonId) { return GetPress(1ul << (int)buttonId); }
public bool GetPressDown(EVRButtonId buttonId) { return GetPressDown(1ul << (int)buttonId); }
public bool GetPressUp(EVRButtonId buttonId) { return GetPressUp(1ul << (int)buttonId); }

public bool GetTouch(ulong buttonMask) { Update(); return (state.ulButtonTouched & buttonMask) != 0; }
public bool GetTouchDown(ulong buttonMask) { Update(); return (state.ulButtonTouched & buttonMask) != 0 &&
(prevState.ulButtonTouched & buttonMask) == 0; }
public bool GetTouchUp(ulong buttonMask) { Update(); return (state.ulButtonTouched & buttonMask) == 0 &&
(prevState.ulButtonTouched & buttonMask) != 0; }

public bool GetTouch(EVRButtonId buttonId) { return GetTouch(1ul << (int)buttonId); }
public bool GetTouchDown(EVRButtonId buttonId) { return GetTouchDown(1ul << (int)buttonId); }
public bool GetTouchUp(EVRButtonId buttonId) { return GetTouchUp(1ul << (int)buttonId); }

                public Vector2 GetAxis(EVRButtonId buttonId = EVRButtonId.k_EButton_SteamVR_Touchpad)
                {
                        Update();
                        var axisId = (uint)buttonId - (uint)EVRButtonId.k_EButton_Axis0;
                        switch (axisId)
                        {
                                case 0: return new Vector2(state.rAxis0.x, state.rAxis0.y);
                                case 1: return new Vector2(state.rAxis1.x, state.rAxis1.y);
                                case 2: return new Vector2(state.rAxis2.x, state.rAxis2.y);
                                case 3: return new Vector2(state.rAxis3.x, state.rAxis3.y);
                                case 4: return new Vector2(state.rAxis4.x, state.rAxis4.y);
                        }
                        return Vector2.zero;
                }
```

Tenemos que decir que en un principio resulta un poco difícil comprender **como acceder a cada byte** del campo de datos (0xB4), ya que la documentación no resulta demasiado clara. Si, como ya hemos comentado anteriormente, suponemos que los **campos de datos del Tracker son los mismo que los de los Controllers** (mandos) empezamos a ver la luz.

Cuando decidimos en nuestro código de Arduino que bytes íbamos a utilizar para mandar los valores recogidos por nuestra botonera, lo hicimos de la siguiente manera:

- Utilizamos el byte para almacenar el PadX para mandar los primeros 15 botones
- Utilizamos el byte destinado a enviar los valores de botón para el numero 16

Por lo tanto si ahora queremos acceder a ese valor de **PadX**, tendremos que usar el mismo método que usaríamos para tomar los valores del **Pad Tactil de los Mandos** (Controlers). Utilizamos una estructura de tipo **Vector2** ya que esta nos permite representar tanto posiciones 2D como vectores.

```
Vector2 touchpad= (device.GetAxis(Valve.VR.EVRButtonId.k_EButton_Axis0));
```

En la siguiente instrucción podemos ver como leemos ese botón que nos hemos visto obligados a mandar a través del byte destinado a botones por falta de espacio en los bytes del PadX. **¿pero los eventos de que botón miramos?**

```
uint8_t buttons = arrayValores [15]; /*Utilizamos el campo buttons para
                                       mandar el bit que no entra en el PadX*/
```

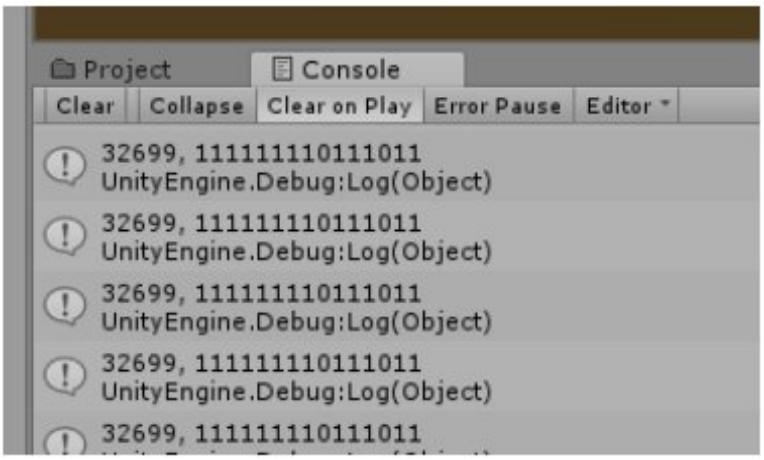
Como podemos ver en la anterior instrucción nosotros **escribimos directamente el valor booleano leido (0/1) en el campo button**. Lo que significa que cuando nuestro microcontrolador lee una pulsación simplemente escribira un 1 en el byte destinado a botones.

TRIGGER	0x01
BUMPER	0x02
MENU	0x04
STEAM	0x08
PAD	0x10
PAD_FINGERDOWN	0x20
Reserved	0x40
Reserved	0x80

Si miramos la tabla cuando el Tracker recibe un **0x01** (un 1 en hexadecimal) determina que el botón de **Trigger** ha sido pulsado como podemos ver en la imagen inferior.

```
ParoEmergencia = device.GetPress(Valve.VR.EVRButtonId.k_EButton_SteamVR_Trigger);
```

Si printeamos los valores recibidos en la consola de Unity, veriamos algo como lo siguiente.



Comos se puede apreciar poco tiene que ver el valor entero entre -32768 y 32767 que "escribiamos" en el Tracker al que recibimos. Esto se debe a que el Tracker mape los valores que nosotros introducimos en el campo PadX para que esten en el **rango entre -1 y 1**. Esto significa que nos tocara a nosotros hacer la operación inversa para obtener los valores originales.

```
int valorPadX = (int) (touchpad.x * 32767);
```

Solo nos queda un ultimo trabajo: volver a convertir el **int** (0-32767) que acabos de conseguir al **array de booleanos** del que partíamos en Arduino. Para esa tarea utilizaremos el metodo ToString que nos permite establecer la base con la que se va a realizar la conversión. Luego dividiremos ese binaryString para guardar cada elemento en una posición del ArrayValorBotones.

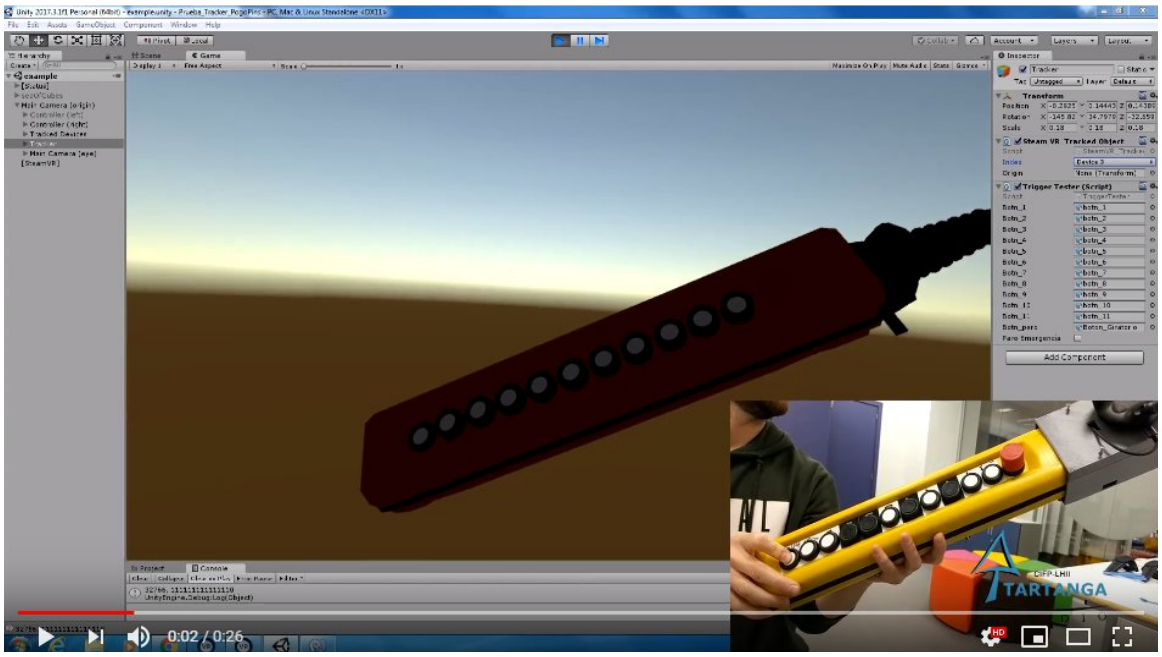
```
string binaryString = Convert.ToString (valorPadX, 2); /*Convertirmos
valorPadX(entero) a un String utilizando base 2*/

bool [] ArrayValorBotones= binaryString.Select(c => c == '0').ToArray();/*Dividimos
el String y almacenamos cada 0/1 en una posición del array*/
```

Hecho esto ya tendremos los valores de los Botones leídos de nuestra botonera en nuestro entorno de desarrollo, fácilmente accesibles en el array que acabamos de crear.

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 mando real.

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.



Link:

https://www.youtube.com/watch?time_continue=1&v=AOyXoBJvv1o&feature=emb_logo