Sistemas Digitales II

Universidad Politécnica de Madrid
Departamento de Ingeniería Electrónica
E.T.S.I. de Telecomunicación

Memoria de las mejoras realizadas

Curso 2018/2019

"PiTankGo: un juguete robótico para la Raspberry Pi"

Autores: Donate Fuentes, Marta

Gómez Rodríguez, Carlos

Código de la pareja: MT-16

ÍNDICE GENERAL

1 Introducción

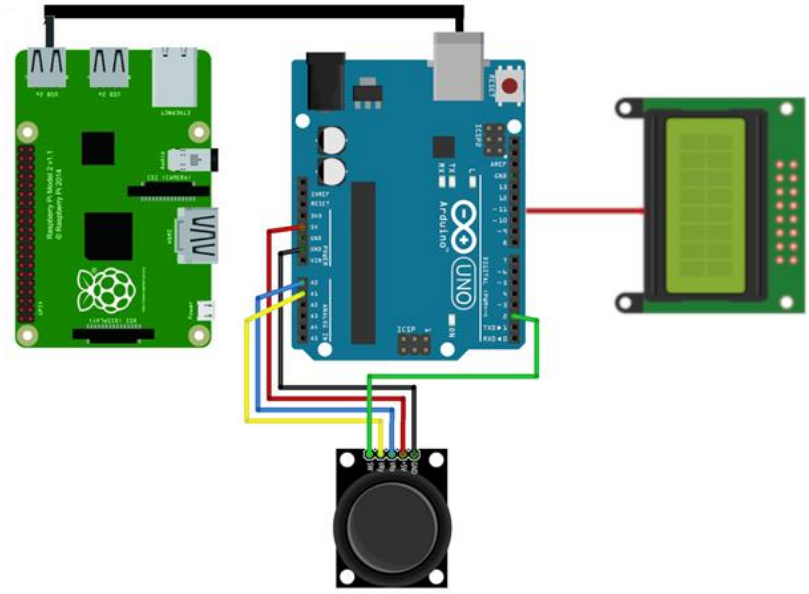
Adicionalmente a la práctica básica hemos incluido unas mejoras que añaden tanto más realismo al juego como más complejidad. Las mejoras incluidas son en su mayoría hibridas, pues tienen una parte un desarrollo hardware como un desarrollo software. La base de las mejoras es la conexión serie entre un Arduino y una Raspberry. Ambas se mandan información por esta conexión, esto nos permite usar subsistemas analógicos gracias al Arduino y una gran capacidad de programación gracias a la Raspberry. Aprovechando las capacidades de los puertos analógicos del Arduino hemos usado un joystick analógico. También hemos incorporado una pantalla LCD 16x2.
Cada una de las mejoras comentadas anteriormente cumplen una función diferente o complementan la versión básica del proyecto:

- Joystick: La práctica se centra en mover una torreta. La propuesta de la práctica básica es usar un teclado matricial para recrear el movimiento de la torreta. Usar un joystick le da un enfoque más realista al proyecto.

- Conexión serie: En objetivo de la asignatura mucha de las cosas enseñadas en Sistemas Digitales I, interrupciones, temporizadores… Sin embargo no es objeto básico de SDG2 la transmisión serie ya que requiere de otra Raspberry o un Arduino. Usando la librería wiringPiSerial hemos podido implementar esta mejora.

- LCD: La información de que se el programa se ejecuta de forma correcta es chequeado por medio del ordenador, pero el objetivo de la práctica es que la Raspberry pueda funcionar correctamente sin necesidad de tener un ordenador o un monitor conectado para chequear como se está desarrollando el programa. Gracias a los mensajes que salen por el LCD podemos seguir los diferentes pasos que nuestro proyecto va realizando.

2 Diagrama de subsistemas ampliado



Existen otros subsistemas como teclado propio, aquí únicamente se han puesto los más importantes. La Raspberry está conectada en serie con un Arduino Mega y a su vez este este conectado por medio de los pines a un teclado LCD 16x2 y un joystick analógico.

A nivel de **software** no ha habido grandes cambios porque se han añadido cosas que no estaban y las que estaban no se han cambiado. Los mayores cambios son la adaptación de la torreta a los datos que llegan por el puerto serie y el hecho de mandar mensajes desde la Raspberry.

Sin embargo se han añadido cosas como el código en Arduino, el cual está creado desde cero. Este código lee el puerto serie, los datos del joystick y escribe en el LCD.

Tras el resumen general de las diferentes mejoras implementadas y el diagrama, se procederá a documentar y detallar individualmente cada mejora específica en un apartado independiente.

3 Mejora 1: Joystick Analógico

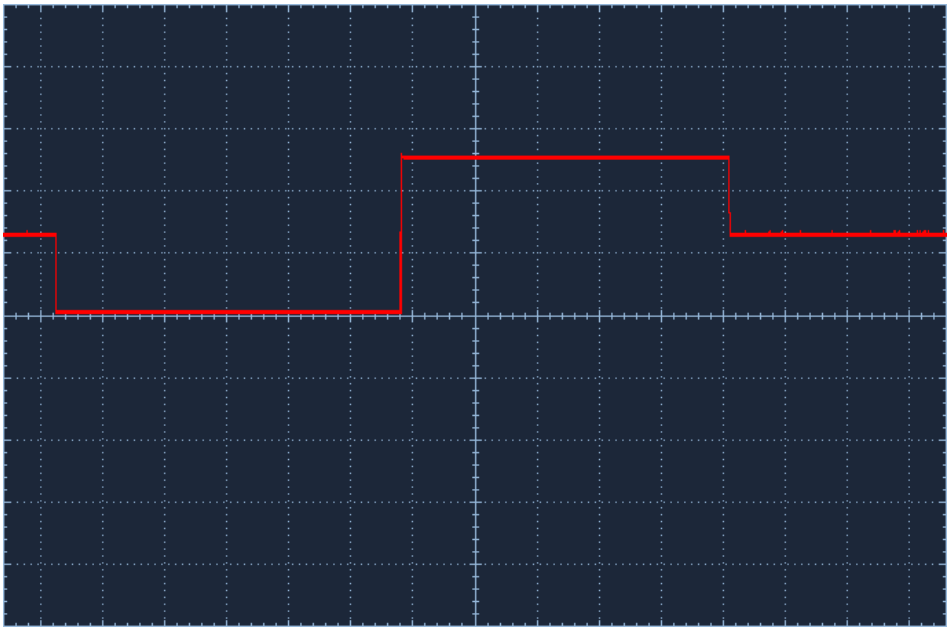
3.1 Objetivos de la mejora

Al tener que mover una torreta física, queda mucho más realista usar un joystick que un teclado matricial. El objetivo es controlar los movimientos de la torreta usando el joystick. Los movimientos a programar son UP, DOWN, LEFT, RIGTH, SHOOT.

3.2 Descripción del subsistema Hardware

El Joystick tiene 3 salidas de datos. Dos son los dos ejes y el tercero es el pulsador. Las dos primeras son analógicas, dan un valor entre 0-5 V. Mientras que el pulsado es digital. Como hay salidas analógicas hacer falta un conversor A/D, para ello usaremos un Arduino que manda un mensaje a la Raspberry según el suceso que se haya producido en el Joystick.

Los ejes del joystick son dos potenciómetros cuyo resultado da un valor analógico entre 0 y 5 V. Cuando el eje está en reposo el valor es 2.5V, justo la mitad. Cuando se echa a un lado es 5V y cuando se echa al otro prácticamente 0V.



Esta foto corresponde a las medidas en el osciloscopio de uno de los ejes (eje x), las divisiones son de 1 V cada cuadricula, en un primer momento se ve 2.5 V (sin evento) después 0 V (izquierda), después 5 V (derecha) y después 2.5 V (sin evento).

Pines

El joystick tiene 5 pines, los pines de 5V y Ground se conectan a los respectivos pines del Arduino. Los pines de medidas analógicas (Dirección X e Y) se conectan a los pines analógicos del Arduino mientras que el pin de pulsado se conecta a un pin digital que se configura como input. Hay que tener en cuenta que el pin de disparo da un valor a nivel bajo, es decir, cuando el botón está pulsado recibimos un

'0' lógico y cuando está pulsado recibimos un '1'. Esto se puede solucionar a nivel software o a nivel hardware (con componentes analógicos). Hemos decidió hacerlo a nivel software por su menor complejidad.

3.3 Descripción del subsistema Software

Usando estos umbrales mandaremos a la Raspberry por el puerto serial (explicado en mejora 2) que suceso se ha producido en el joystick. Los eventos se cifran de la siguiente forma:

```
#define BUTTON    01 || %001
#define UP        02 || %010
#define DOWN      03 || %011
#define LEFT      04 || %100
#define RIGHT     05 || %101
```

Esta información será la que mandemos por el puerto serie.

Flujo de ejecución del programa principal

Diagrama de Flujo:

1) Consultamos las medidas analógicas.
2) Mediante umbrales sacamos el evento en el Joystick.
3) Mandamos la información por el puerto serie (Mejora 2).

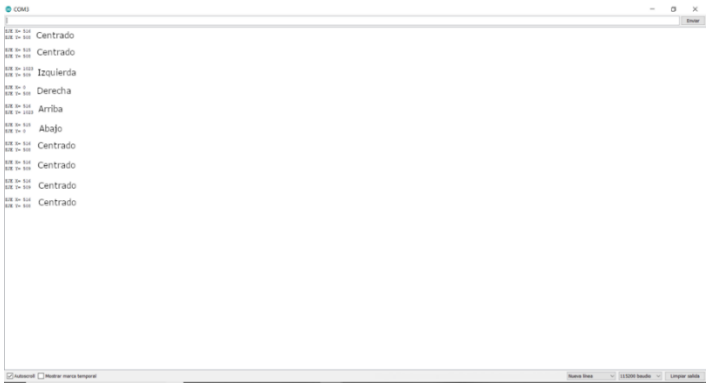
- Justificación de la solución adoptada.

La información se cifra en 3 bits. Esto hace que no se pueda ir en dos posiciones a la vez (por ejemplo activar al mismo tiempo UP-LEFT). Si usáramos 5 bits para cifrar la información no habría problema. Se ha adoptado esta solución debido a que a la hora de transmitir los datos por el puerto serie los bits de información estaban limitados a 3.

Las variables de entrada son 3 enteros, los dos ejes y el botón. Como variable de salida tenemos un único entero en el que está toda la información codificada. También para evitar que se envie dos veces la misma información hay variables que guardan el último valor que se ha enviado y si es el mismo que el que se va a enviar no lo envía para así no enviar datos duplicados fruto de ruido.

Joystick al Arduino

El objetivo es a través de código transformar un joystick analógico en un joystick digital, estos datos pasárselos a la Raspberry y que otro subsistema se encargue de procesar la información. El Arduino lee los datos de los pines que se estén usando. En el caso de las medidas analógicas discretiza 0-5V en 0 a 1223.



Cuando el joystick está en reposo se lee 500, cuando se mueve a uno de los lados 0 y cuando se mueve al otro 1000 por lo que habrá que configurar los umbrales sobre 250 y 750.

Arduino a Raspberry

En este apartado no se va a explicar como se envía la información ya que la comunicación serie es otro subsistema si no que se va a explicar como se codifica.

```
#define BUTTON    01 || %001
#define UP        02 || %010
#define DOWN      03 || %011
#define LEFT      04 || %100
#define RIGHT     05 || %101
```

Usamos esta tabla para codificar en un entero los posibles eventos. En la Raspberry de forma análoga con un switch (int x) se podrán diferentes cases para las combinaciones que hemos programado.

Procesos de las interrupciones

El joystick cambia 5 flags de la torreta. Estos flags son las que activan las diferentes transiciones de la máquina de estados de la torreta.

4 Mejora 2 : Conexión Serie

4.1 Objetivos de la mejora

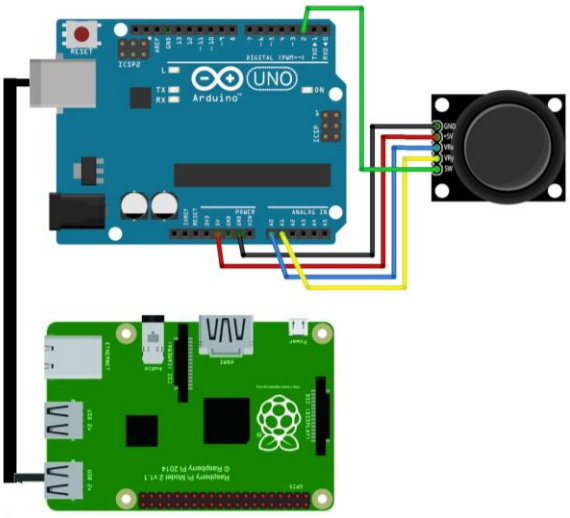
Anteriormente la conexión entre el Arduino y la Raspberry para usar el joystick era una conexión con 5 cables a interrupciones y un puente de masas. El problema de esto era la necesidad de un gran número de cables. La conexión serie es mucho mejor porque además de ser a nivel de hardware más sencilla, se puede mandar mucha más información que con interrupciones. Esta es la mejora más importante ya que sin ella las otras dos no serías posibles o habría que modificarlas.

4.2 Descripción del subsistema Hardware

Tras una breve descripción de los módulos que forman el subsistema Hardware requerido por la mejora se procederá a describir cada módulo Hardware en apartados diferentes.

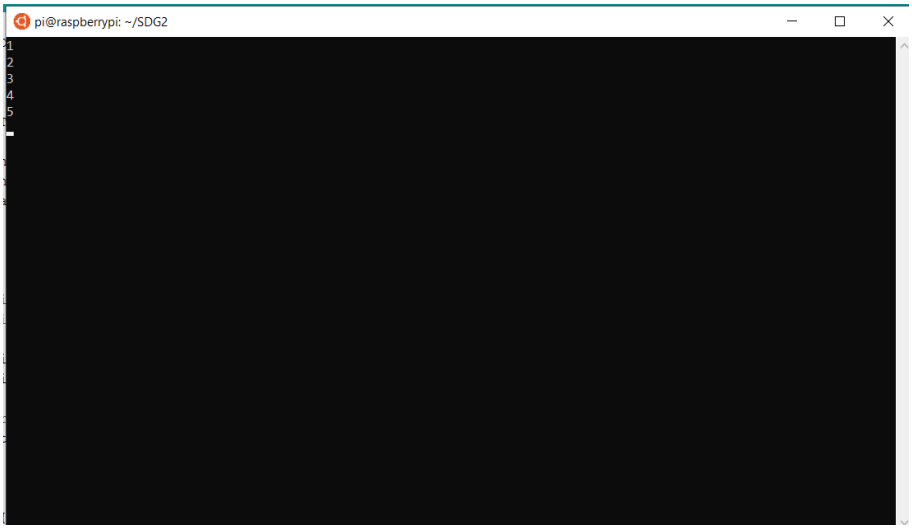
Conexión

La conexión serie se basa en el uso únicamente de 3 cables. Hay que conectar el pin de TX del Arduino al pin de RX de la Raspberry y el pin de RX del Arduino al pin de TX de la Raspberry y cortocircuitar las masas; con el cable USB además de estos tres cables también se alimenta la Raspberry, pero la metodología explicada anteriormente se respeta.



4.3 Descripción del subsistema Software

Gracias a la herramienta screen podemos ver directamente lo que se recibe por el puerto serial en la Raspberry. En este caso se puede ver que se han mandado los 5 eventos del joystick codificados según se ha explicado en la mejora anterior.



Flujo de ejecución del programa principal

La comunicación serie se divide en dos subsistemas, en el primero la Raspberry manda datos y el Arduino los recibe y en el otro análogamente.

Comunicación Arduino-Raspberry

Fue la primera que se implementó y la más completa. **Para entender la solución adoptada en el código es conveniente leer antes el punto 5) de problemas.**

Esta vía de comunicación es muy compleja. Aunque a simple vista parece que todo funciona perfectamente, en el código compilado en C no es así y los métodos de wiring Pi no son capaces de extraer la información útil de los bits de parada, paridad…

La información recibida eran 3 enteros y no se sabía en cual iba la información útil (codificada de la mejora 1). Finalmente vimos que cuando se mandaban eventos diferentes únicamente cambiaban las unidades (variaciones de <10) del primer entero. Analizando el problema aun a mayor profundidad vemos que lo único que cambia son los 3 bits menos significativos; por lo que usando una máscara en el primer entero de los 3 bits menos significativos (int x &=7) podemos extraer la información útil.

Procesos de las interrupciones

El dispositivo más comprometido es el que recibe, ya que se ve obligado a hacer un escaneo constante del puerto por el que recibe para comprobar si está recibiendo información

Comunicación Arduino-Raspberry

El código en la Raspberry es muy complejo ya que está todo el rato recorriendo las tabla de transición de las máquinas de estado y sus diferentes métodos. La mejor solución es crear una hebra independiente que se dedique únicamente a leer el puerto serie. Al tratarse de una hebra independiente no hace falta usar interrupciones. Esta hebra interactúa con la hebra principal (main) activando los flags de la máquina de estados de la torreta. Esto provocará distintas transiciones en las máquinas de estados.

Comunicación Raspberry-Arduino

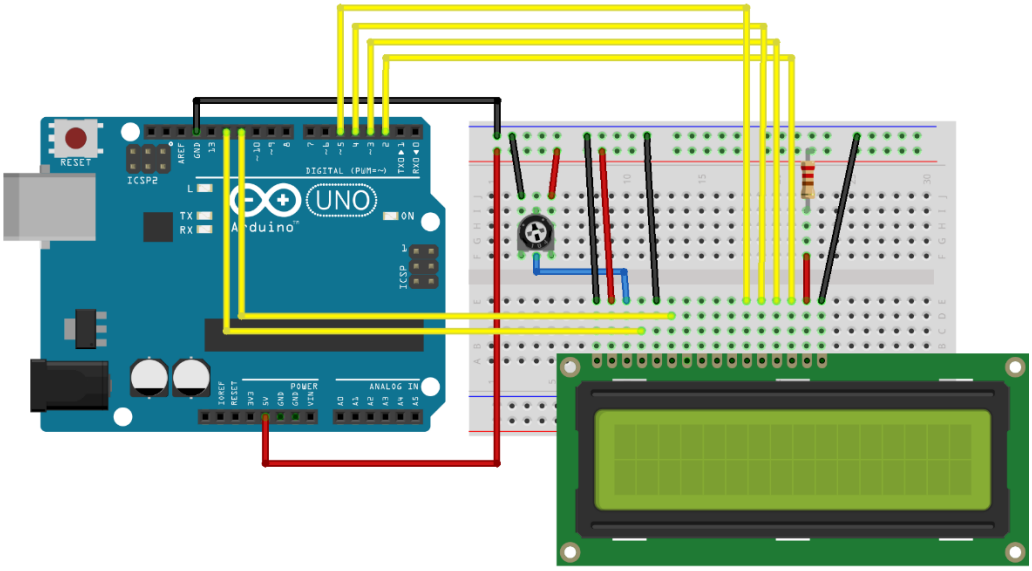
En Arduino no se pueden crear varias hebras así que no se puede usar la misma estrategia usada anteriormente. Sin embargo, el programa en Arduino es muy simple y el hacer una lectura constante del puerto serie es viable.

5 Mejora 3: Pantalla LCD

5.1 Objetivos de la mejora

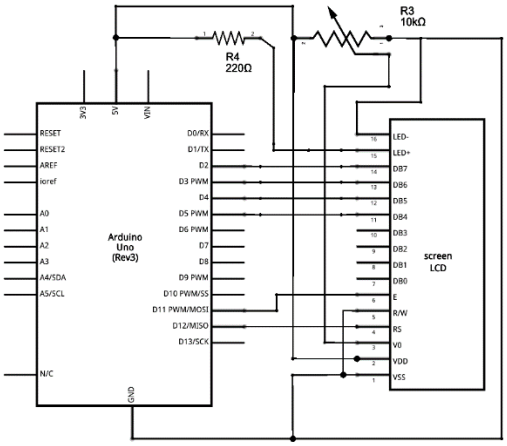
El objetivo es prescindir completamente de la pantalla del ordenador para que toda la información salga por una pantalla LCD. Igual que en la mejora anterior, se recomienda leer antes en la sección de problemas el punto 6)

5.2 Descripción del subsistema Hardware



El esquema de Hardware que se ha usado es el siguiente. A continuación se describirá que función cumple cada una de las conexiones

Descripción de los pines



Los pines se van a describir de izquierda a derecha:

-VSS: Ground.
-VDD: +5V.
-V0: Control del brillo por medio de la tensión que llega al pin.
-RS: Selección de registro de control de datos (0) o registro de datos (1) (Conectado al Arduino).
-R/W: Lectura/Escritura, conectado a Gound (0V).
-E: Enable (Conectado al Arduino).
-DB0-DB3: DataPins, sin conectar.
-DB4-DB7: DataPins (Conectados al Arduino).
-Led+: Conectado a una Resistencia conectada a 5V.
-Led-: Conectada a Ground (0V).

Control del brillo

Tal y como se aparece en el diagrama de arriba, es necesario con un potenciómetro ajustar el brillo de la pantalla LCD.

5.3 Descripción del subsistema Software

Gracias a las librerías disponibles para Arduino el uso del LCD se simplifica muchísimo.

Flujo de ejecución del programa principal

Existen dos subrutinas diferentes, por una parte, van llegando mensajes sobre los eventos de la torreta que el LCD va sacando por pantalla, por otra el LCD tiene un marcador que saca las veces que tras un disparo se ha producido un impacto.

Eventos

Cuando llega un mensaje nuevo que es diferente del último que se ha recibido, el LCD se actualiza y saca por pantalla dicho mensaje.

Marcador

La intención era mandar un entero actualizado con el número de impactos cada vez que se produjese uno, pero en Arduino no se puede distinguir un String de un entero; por ello la solución implementada ha sido mandar una "x" cada vez que se produce un impacto y es el propio Arduino quien hace el incremento.

6 Mejora 4: Receptor Óptico

Usando el cañón del laboratorio tenemos un receptor para que se produzca el impacto

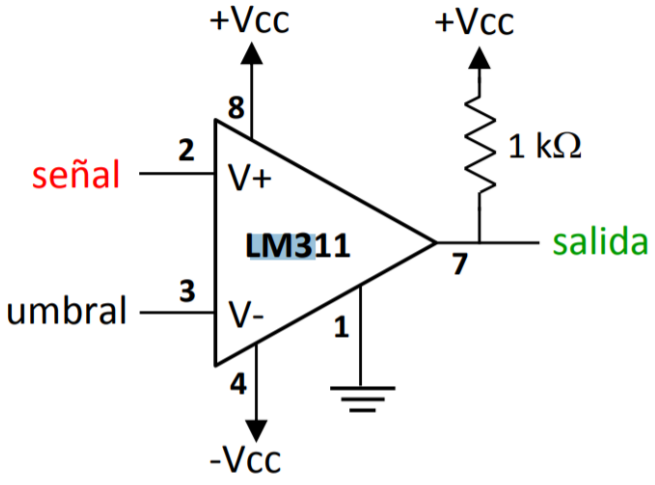
6.1 Objetivos de la mejora

El receptor está conectado a un pin de la Raspberry que produce la interrupción de Target detectado.

6.2 Descripción del subsistema Hardware

Con el circuito propuesto en el enunciado no conseguíamos llegar a un volteje alto cuando el fotodiodo recibía luz así que hemos recurrido a un montaje alternativo.

Circuito



A este circuito hay que conecta a la pata señal el fotodiodo que su otra pata estará conectada a Vcc (hay que tener cuidado con la polarización) y una resistencia que estará conectada a Masa. A la pata umbral conectaremos a la pata central de un potenciómetro, sus otras dos patas estarán conectadas a Vcc y masa.

El funcionamiento del circuito es muy simple. Siempre que la pata señal haya más voltaje que en la pata umbral la salida serán 5 V. Experimentalmente se ha comprobado que cuando hay fotocorriente el voltaje es de unos 5 V así que colocaremos el potenciómetro para que en la pata umbral haya alrededor de 3 V.

7 Principales problemas encontrados

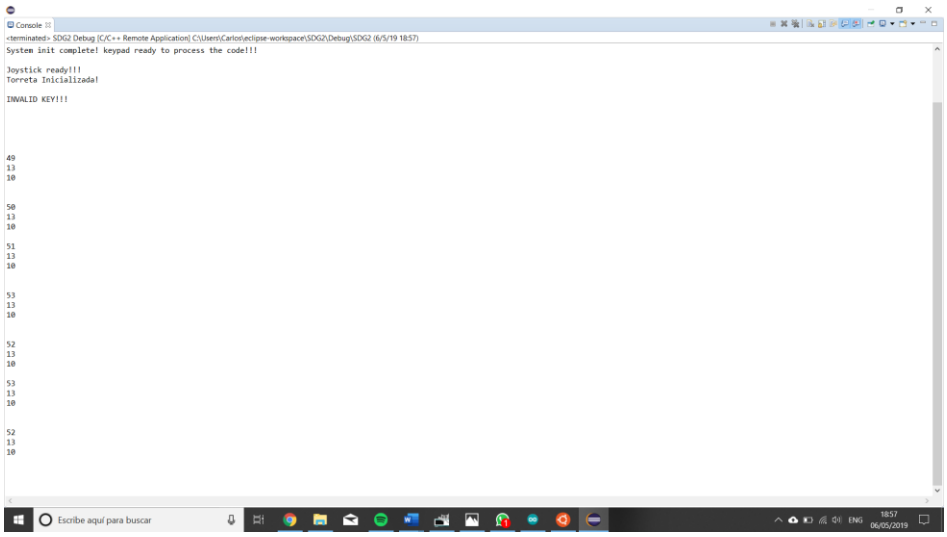
1) Debido a que el cable de alimentación que tenemos da una gran corriente y a un error a la hora de conectar el servo hizo que la Raspberry se rompiese y no llegase a arrancar ni el S.O.

2) Para la conexión serie entre el Arduino y la Raspberry, el objetivo era usar los pines de TX, RX y Ground pero no conseguíamos mandar datos, por ello se decidió establecer la comunicación serie con el cable USB que no dio problemas.

3) Cuando el Arduino y la Raspberry estaban conectador con los 5 pines que emulaban el joystick digital, había un error porque había que hacer un puente de masas entre ambos dispositivos.

4) Al usar Raspberry propia y desarrollar el proyecto en el ordenador personal es necesario configurar la conexión desde 0 entre ambos dispositivos. Al principio usábamos el móvil como un punto de red wifi al que estaban conectados ambos dispositivos, aunque debido a la necesidad de un aparato más, finalmente se decidió usar un cable Ethernet entre ambos dispositivos.

5) Al implementar la conexión serial, en la herramienta screen no había ningún problema, pero en el programa en C se imprimía toda la ristra de bits, incluyendo paridad, parada… La solución adoptada para solucionar el problema se ha explicado en la mejora 2.



6) La idea inicial era conectar la pantalla LCD a la Raspberry Pi pero debido a que la Raspberry dispone de pocos pines GPIO consideramos que era más optimo conectar la pantalla al Arduino y mandar los datos a imprimir por pantalla por la conexión serie que ya estaba creada. Gracias a la librería <LiquidCrystal.h> podemos usar el LCD de una forma más fácil.

8 Manual de usuario

El objetivo de este proyecto ha sido tanto el realismo de la torreta como la independencia del programa del PC. La primera fase fue el manejo de la torreta desde un teclado matricial para así no tener que usar el teclado del ordenador. En una segunda fase puesto que el objetivo es mover una torreta, se usa un joystick como sustituto del teclado. Finalmente se añade una pantalla por la que van saliendo los mensajes para que no sea necesario recurrir a la pantalla del ordenador.

Uso del proyecto:

1) Tenemos un Joystick con el que se puede controlar el brazo de la torreta. Cada unos lados del joystick mueve la torreta a un lado diferente. Al pulsar el botón central del Joystick se activa el disparo con un temporizador que mantendrá activo el disparo durante un tiempo fijado.

2) En caso de querer usar el teclado matricial. Se ha seguido la siguiente combinación de teclas:

-Tecla ‘2’: UP.

-Tecla ‘8’: DOWN.

-Tecla ‘4’: LEFT.

-Tecla ‘6’: RIGTH.

-Tecla ‘5’: DISPARO.

-Tecla ‘0’: IMPACTO.

La tecla ‘0’ se ha añadido para que en caso de que no se produjese la interrupción del receptor comprobar el correcto funcionamiento del código.

3) Hay una pantalla LCD que va sacando por los diferentes eventos que se van produciendo en la torreta además de un marcador que lleva la cuenta de los Impactos producidos.

COMO CARGAR EL PROGRAMA:

1. Cargar el fichero arduino_serie.ino que se encuentra dentro del zip y en la carpeta Arduino en el Arduino, para ello abrimos el programa de Arduino y compilamos el programa. Una vez el programa esté dentro cada vez que se alimente el Arduino el programa se ejecutara directamente como si al encender un ordenador se tratase del sistema operativo.

2. Una vez cargado el programa introducimos el USB en la Raspberry. Abrimos el programa en Eclipse y ejecutamos el programa con el compilador cruzado.

3. Una vez el programa esté dentro de nuestra Raspberry podremos acceder a él a través de una bash de Unix accediendo a la carpeta que esté el fichero (Se elegía en las opciones de Eclipse) y ejecutamos el comando: ./NOMBE_DEL_PROGRAMA.

9 Bibliografía

Datasheet Joystick
http://www.energiazero.org/arduino_sensori/joystick_module.pdf

Datasheet Teclado Matricial 4x4
https://www.luisllamas.es/arduino-teclado-matricial/

Datasheet LCD 16x2
https://www.sparkfun.com/datasheets/LCD/ADM1602K-NSW-FBS-3.3v.pdf

Ejemplo “Hello World”LCD 16x2
https://www.arduino.cc/en/Tutorial/HelloWorld

Librería Wiring Pi Serial
http://wiringpi.com/reference/serial-library/

10 ANEXO I: Código del programa del proyecto final

A continuación se suben los dos códigos principales y los que se han visto en mayor parte alterados por las mejoras.

```
/** File Name          : arduino_seria.ino
  * Description        : Programa para el arduino
  */


#include <SoftwareSerial.h>
#include <LiquidCrystal.h>

const int rs = 7, en = 6, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

//PINES SALIDA
#define xPin A14
#define yPin A15
#define buttonPin 52

#define BUTTON     01 //0 1
#define UP         02 //1 2
#define DOWN       03 //2 4
#define LEFT       04 //3 8
#define RIGHT      05 //4 16

int xPosition = 0;
int yPosition = 0;
int buttonState = 0;
int var = 0;
int last_var =0;
int marcador = 0;
 String mensaje;
String lastmensaje;
void setup() {
  // inicializar las comunicaciones en serie a 112500 bps:
  Serial.begin(115200);
  lcd.begin(16, 2);

  pinMode(xPin, INPUT);
  pinMode(yPin, INPUT);
  //activar resistencia pull-up en el pin pulsador
```

```
  pinMode(buttonPin, INPUT_PULLUP);

  lcd.setCursor(15,1);
  lcd.print(marcador);

}

void loop() {
  xPosition = analogRead(xPin);
  yPosition = analogRead(yPin);
  buttonState = digitalRead(buttonPin);


  //RIGHT
  if(xPosition>600){
    var=RIGHT;
    }
   //LEFT
   if(xPosition<250){
    var=LEFT;   }
  //UP
   if(yPosition>750){
    var=UP;
    }

  //DOWN
  if(yPosition<250){
    var=DOWN;
    }

  //BUTTON
  if(buttonState==0){
    var=BUTTON;
    }


  if(var!=last_var && var!=0){
  Serial.println(var);
  delay(300);
  var=0;
  }
  if(Serial.available()>0){

    lcd.clear();
```

```
    mensaje= Serial.readString();

    if (mensaje=="x"){
       marcador++;
    }else{
      lastmensaje=mensaje;
    }
       lcd.setCursor(0,0);
       lcd.print(lastmensaje);
       lcd.setCursor(15,1);
       lcd.print(marcador);

    delay(300);
  }

}
```

```
/** File Name          : piTankGo_1.c
  * Description        : Programa principal
  */

//Librerias necesarias
#include "piTankGo_1.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <wiringPi.h>
#include <wiringSerial.h>
#include <softTone.h>
#include <pthread.h>
#include "fsm.h"
#include "tmr.h"
#include "teclado.h"
#include "kbhit.h"
#include "piTankGoLib.h"

int frecuenciaDespacito[160] = { 0, 1175, 1109, 988, 740, 740, 740, 740, 740,
              740, 988, 988, 988, 988, 880, 988, 784, 0, 784, 784, 784, 784, 784, 988,
              988, 988, 988, 1109, 1175, 880, 0, 880, 880, 880, 880, 880, 1175, 1175,
              1175, 1175, 1318, 1318, 1109, 0, 1175, 1109, 988, 740, 740, 740, 740,
```

```
              740, 740, 988, 988, 988, 988, 880, 988, 784, 0, 784, 784, 784, 784, 784,
              988, 988, 988, 988, 1109, 1175, 880, 0, 880, 880, 880, 880, 880, 1175,
              1175, 1175, 1175, 1318, 1318, 1109, 0, 1480, 1318, 1480, 1318, 1480,
              1318, 1480, 1318, 1480, 1318, 1480, 1568, 1568, 1175, 0, 1175, 1568,
              1568, 1568, 0, 1568, 1760, 1568, 1480, 0, 1480, 1480, 1480, 1760, 1568,
              1480, 1318, 659, 659, 659, 659, 659, 659, 659, 659, 554, 587, 1480,
              1318, 1480, 1318, 1480, 1318, 1480, 1318, 1480, 1318, 1480, 1568, 1568,
              1175, 0, 1175, 1568, 1568, 1568, 1568, 1760, 1568, 1480, 0, 1480, 1480,
              1480, 1760, 1568, 1480, 1318 };
int tiempoDespacito[160] = { 1200, 600, 600, 300, 300, 150, 150, 150, 150, 150,
              150, 150, 150, 300, 150, 300, 343, 112, 150, 150, 150, 150, 150, 150,
              150, 150, 300, 150, 300, 300, 150, 150, 150, 150, 150, 150, 150, 150,
              150, 300, 150, 300, 800, 300, 600, 600, 300, 300, 150, 150, 150, 150,
              150, 150, 150, 150, 300, 150, 300, 343, 112, 150, 150, 150, 150, 150,
              150, 150, 150, 300, 150, 300, 300, 150, 150, 150, 150, 150, 150, 150,
              150, 150, 300, 150, 300, 450, 1800, 150, 150, 150, 150, 300, 150, 300,
              150, 150, 150, 300, 150, 300, 450, 450, 300, 150, 150, 225, 75, 150,
              150, 300, 450, 800, 150, 150, 300, 150, 150, 300, 450, 150, 150, 150,
              150, 150, 150, 150, 150, 300, 300, 150, 150, 150, 150, 150, 150, 450,
              150, 150, 150, 300, 150, 300, 450, 450, 300, 150, 150, 150, 300, 150,
              300, 450, 800, 150, 150, 300, 150, 150, 300, 450 };
int frecuenciaGOT[518] = { 1568, 0, 1046, 0, 1244, 0, 1397, 0, 1568, 0, 1046, 0,
              1244, 0, 1397, 0, 1175, 0, 1397, 0, 932, 0, 1244, 0, 1175, 0, 1397, 0,
              932, 0, 1244, 0, 1175, 0, 1046, 0, 831, 0, 698, 0, 523, 0, 349, 0, 784,
              0, 523, 0, 523, 0, 587, 0, 622, 0, 698, 0, 784, 0, 523, 0, 622, 0, 698,
              0, 784, 0, 523, 0, 622, 0, 698, 0, 587, 0, 698, 0, 466, 0, 622, 0, 587,
              0, 698, 0, 466, 0, 622, 0, 587, 0, 523, 0, 523, 0, 587, 0, 622, 0, 698,
              0, 784, 0, 523, 0, 622, 0, 698, 0, 784, 0, 523, 0, 622, 0, 698, 0, 587,
              0, 698, 0, 466, 0, 622, 0, 587, 0, 698, 0, 466, 0, 622, 0, 587, 0, 523,
              0, 0, 1568, 0, 0, 1046, 0, 0, 1244, 0, 0, 1397, 0, 0, 1568, 0, 0, 1046,
              0, 0, 1244, 0, 0, 1397, 0, 0, 1175, 0, 587, 0, 622, 0, 587, 0, 523, 0,
              587, 0, 784, 0, 880, 0, 932, 0, 1046, 0, 1175, 0, 0, 1397, 0, 0, 932, 0,
              0, 1244, 0, 0, 1175, 0, 0, 1397, 0, 0, 932, 0, 0, 1244, 0, 0, 1175, 0,
              0, 1046, 0, 0, 1568, 0, 0, 1046, 0, 0, 1244, 0, 0, 1397, 0, 0, 1568, 0,
              0, 1046, 0, 0, 1244, 0, 0, 1397, 0, 0, 1175, 0, 880, 0, 784, 0, 932, 0,
              1244, 0, 0, 1397, 0, 0, 932, 0, 0, 1175, 0, 0, 1244, 0, 0, 1175, 0, 0,
              932, 0, 0, 1046, 0, 0, 2093, 0, 622, 0, 831, 0, 932, 0, 1046, 0, 622, 0,
              831, 0, 1046, 0, 0, 1865, 0, 622, 0, 784, 0, 831, 0, 932, 0, 622, 0,
              784, 0, 932, 0, 0, 1661, 0, 523, 0, 698, 0, 784, 0, 831, 0, 523, 0, 698,
              0, 831, 0, 0, 1568, 0, 1046, 0, 1244, 0, 1397, 0, 1568, 0, 1046, 0,
              1244, 0, 1397, 0, 0, 0, 1661, 0, 1046, 0, 1175, 0, 1244, 0, 831, 0,
              1175, 0, 1244, 0, 0, 0, 0, 2489, 0, 0, 0, 0, 2794, 0, 0, 0, 0, 3136, 0,
              0, 2093, 0, 622, 0, 831, 0, 932, 0, 1046, 0, 622, 0, 831, 0, 1046, 0, 0,
              1865, 0, 622, 0, 784, 0, 831, 0, 932, 0, 622, 0, 784, 0, 932, 0, 0,
```

```
              1661, 0, 523, 0, 698, 0, 784, 0, 831, 0, 523, 0, 698, 0, 831, 0, 0,
              1568, 0, 1046, 0, 1244, 0, 1397, 0, 1568, 0, 1046, 0, 1244, 0, 1397, 0,
              0, 0, 1661, 0, 1046, 0, 1175, 0, 1244, 0, 831, 0, 1175, 0, 1244, 0, 0,
              0, 0, 2489, 0, 1397, 0, 0, 0, 2350, 0, 0, 0, 2489, 0, 0, 0, 2350, 0, 0,
              0, 0, 2093, 0, 392, 0, 415, 0, 466, 0, 523, 0, 392, 0, 415, 0, 466, 0,
              523, 0, 392, 0, 415, 0, 466, 0, 2093, 0, 1568, 0, 1661, 0, 1865, 0,
              2093, 0, 1568, 0, 1661, 0, 1865, 0, 2093, 0, 1568, 0, 1661, 0, 1865 };
int tiempoGOT[518] = { 900, 89, 900, 89, 133, 13, 133, 13, 600, 59, 600, 59,
              133, 13, 133, 13, 1400, 1400, 900, 89, 900, 89, 133, 13, 133, 13, 600,
              59, 900, 89, 133, 13, 133, 13, 1200, 116, 267, 28, 267, 28, 267, 28,
              900, 89, 900, 89, 1400, 89, 69, 7, 69, 7, 69, 7, 69, 7, 900, 89, 900,
              89, 133, 13, 133, 13, 600, 59, 600, 59, 133, 13, 133, 13, 1800, 1800,
              900, 89, 900, 89, 133, 13, 133, 13, 600, 59, 900, 89, 133, 13, 133, 13,
              1200, 2400, 69, 7, 69, 7, 69, 7, 69, 7, 900, 89, 900, 89, 133, 13, 133,
              13, 600, 59, 600, 59, 133, 13, 133, 13, 1800, 1800, 900, 89, 900, 89,
              133, 13, 133, 13, 600, 59, 900, 89, 133, 13, 133, 13, 1200, 2400, 3600,
              900, 89, 900, 900, 89, 900, 133, 13, 150, 133, 13, 150, 600, 59, 600,
              600, 59, 600, 133, 13, 150, 133, 13, 150, 1200, 400, 69, 7, 69, 7, 69,
              7, 69, 7, 267, 28, 400, 45, 133, 13, 267, 28, 267, 28, 267, 28, 300,
              900, 89, 900, 900, 89, 900, 133, 13, 150, 133, 13, 150, 600, 59, 600,
              900, 89, 900, 133, 13, 150, 133, 13, 150, 1200, 1800, 3600, 900, 89,
              900, 900, 89, 900, 133, 13, 150, 133, 13, 150, 600, 59, 600, 600, 59,
              600, 133, 13, 150, 133, 13, 150, 1200, 400, 267, 28, 1200, 400, 133, 13,
              133, 13, 150, 900, 89, 900, 900, 89, 900, 600, 59, 600, 267, 28, 300,
              600, 59, 600, 267, 28, 300, 1200, 2400, 3600, 267, 28, 267, 28, 133, 13,
              133, 13, 267, 28, 267, 28, 133, 13, 133, 13, 150, 267, 28, 267, 28, 133,
              13, 133, 13, 133, 13, 267, 28, 267, 28, 133, 13, 150, 267, 28, 267, 28,
              133, 13, 133, 13, 267, 28, 267, 28, 133, 13, 133, 13, 150, 267, 28, 267,
              28, 133, 13, 133, 13, 267, 28, 267, 28, 133, 13, 133, 13, 150, 150, 600,
              59, 133, 13, 133, 13, 267, 28, 267, 28, 133, 13, 133, 13, 150, 150, 150,
              900, 89, 900, 900, 900, 900, 89, 900, 900, 900, 1200, 2400, 3600, 267,
              28, 267, 28, 133, 13, 133, 13, 267, 28, 267, 28, 133, 13, 133, 13, 150,
              267, 28, 267, 28, 133, 13, 133, 13, 267, 28, 267, 28, 133, 13, 133, 13,
              150, 267, 28, 267, 28, 133, 13, 133, 13, 267, 28, 267, 28, 133, 13, 133,
              13, 150, 267, 28, 267, 28, 133, 13, 133, 13, 267, 28, 267, 28, 133, 13,
              133, 13, 150, 150, 600, 59, 133, 13, 133, 13, 267, 28, 267, 28, 133, 13,
              133, 13, 150, 150, 150, 600, 212, 133, 13, 150, 150, 267, 28, 300, 300,
              400, 45, 450, 450, 133, 13, 150, 150, 150, 267, 28, 267, 28, 133, 13,
              133, 13, 267, 28, 267, 28, 133, 13, 133, 13, 267, 28, 267, 28, 133, 13,
              2400, 116, 267, 28, 267, 28, 133, 13, 133, 13, 267, 28, 267, 28, 133,
              13, 133, 13, 267, 28, 267, 28, 133, 13, 2400 };
int frecuenciaTetris[55] = { 1319, 988, 1047, 1175, 1047, 988, 880, 880, 1047,
              1319, 1175, 1047, 988, 988, 1047, 1175, 1319, 1047, 880, 880, 0, 1175,
              1397, 1760, 1568, 1397, 1319, 1047, 1319, 1175, 1047, 988, 988, 1047,
```

```
              1175, 1319, 1047, 880, 880, 0, 659, 523, 587, 494, 523, 440, 415, 659,
              523, 587, 494, 523, 659, 880, 831 };
int tiempoTetris[55] = { 450, 225, 225, 450, 225, 225, 450, 225, 225, 450, 225,
              225, 450, 225, 225, 450, 450, 450, 450, 450, 675, 450, 225, 450, 225,
              225, 675, 225, 450, 225, 225, 450, 225, 225, 450, 450, 450, 450, 450,
              450, 900, 900, 900, 900, 900, 900, 1800, 900, 900, 900, 900, 450, 450,
              900, 1800 };
int frecuenciaStarwars[59] = { 523, 0, 523, 0, 523, 0, 698, 0, 1046, 0, 0, 880,
              0, 784, 0, 1397, 0, 523, 0, 1760, 0, 0, 880, 0, 784, 0, 1397, 0, 523, 0,
              1760, 0, 0, 880, 0, 784, 0, 1397, 0, 523, 0, 1760, 0, 0, 880, 0, 1760,
              0, 0, 784, 0, 523, 0, 0, 523, 0, 0, 523, 0 };
int tiempoStarwars[59] = { 134, 134, 134, 134, 134, 134, 536, 134, 536, 134,
              134, 134, 134, 134, 134, 536, 134, 402, 134, 134, 429, 357, 134, 134,
              134, 134, 536, 134, 402, 134, 134, 429, 357, 134, 134, 134, 134, 536,
              134, 402, 134, 134, 429, 357, 134, 134, 134, 429, 357, 1071, 268, 67,
              67, 268, 67, 67, 67, 67, 67 };

int frecuenciasDisparo[16] = { 2500, 2400, 2300, 2200, 2100, 2000, 1900, 1800,
              1700, 1600, 1500, 1400, 1300, 1200, 1100, 1000 };
int tiemposDisparo[16] = { 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75,
              75, 75, 75 };
int frecuenciasImpacto[32] =
              { 97, 109, 79, 121, 80, 127, 123, 75, 119, 96, 71, 101, 98, 113, 92, 70,
                            114, 75, 86, 103, 126, 118, 128, 77, 114, 119, 72 };
int tiemposImpacto[32] = { 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
              10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
              10 };

int flags_torreta = 0;
int flags_player = 0;
int flags_teclado = 0;

//------------------------------------------------------
// FUNCIONES DE CONFIGURACION/INICIALIZACION
//------------------------------------------------------

int ConfiguraSistema(TipoSistema* p_sistema) {

       int result = 0;
       wiringPiSetupGpio();
       pinMode(PLAYER_PWM_PIN, OUTPUT);
       softToneCreate(PLAYER_PWM_PIN);
       p_sistema->player.tmr = tmr_new(timer_player_duracion_nota_actual_isr);
       p_sistema->torreta.tmr = tmr_new(timer_duracion_disparo_isr);
```

```
	return result;
}

int InicializaSistema(TipoSistema *p_sistema) {
	int result = 0;

	//Incializa los diferentes efectos
	InicializaEfecto(&p_sistema->player.efecto_Despacito, "DESPACITO",
			frecuenciaDespacito, tiempoDespacito, 160);
	InicializaEfecto(&p_sistema->player.efecto_GOT, "GOT", frecuenciaGOT,
			tiempoGOT, 518);
	InicializaEfecto(&p_sistema->player.efecto_Tetris, "TETRIS",
			frecuenciaTetris, tiempoTetris, 55);
	InicializaEfecto(&p_sistema->player.efecto_Starwars, "STARWARS",
			frecuenciaStarwars, tiempoStarwars, 59);
	InicializaEfecto(&p_sistema->player.efecto_disparo, "DISPARO",
			frecuenciasDisparo, tiemposDisparo, 16);
	InicializaEfecto(&p_sistema->player.efecto_impacto, "IMPACTO",
			frecuenciasImpacto, tiemposImpacto, 32);


	//Inicializamos las diferentes subrutinas
	InicializaTeclado(&p_sistema->teclado);
	InicializaJoystick(&p_sistema->joystick);
	InicializaTorreta(&p_sistema->torreta);

	// Lanzamos thread para exploracion del teclado del PC
	result = piThreadCreate(thread_explora_teclado_PC);
	if (result != 0) {
		printf("Thread didn't start!!!\n");
		return -1;
	}
	// Lanzamos thread para exploracion del puerto serie
	result = piThreadCreate(thread_UART);
	if (result != 0) {
		printf("Thread didn't start!!!\n");
		return -1;
	}
	return result;
}

PI_THREAD(thread_UART) {
	c = 0;
```

```
	key = 0; //Sirve para no considerar la primera ristra de bits
	//Inicializa el puerto serie a 115200 baudios
	fd =   serialOpen(   "/dev/serial/by-id/usb-Arduino__www.arduino.cc__0042_85734323313519170D1-if00",115200);


	while (1) {
		c = 0;
		while (serialDataAvail(fd)) {
			int m = serialGetchar(fd);
			if (m != -1)
				m &= 7;
			if (c < 1 && key == 1) {
				//Toma una decision dependiendo del evento producido
				/* #define BUTTON    01 || %001
				 * #define UP        02 || %010
				 * #define DOWN      03 || %011
				 * #define LEFT      04 || %100
				 * #define RIGHT     05 || %101
				 */
				if (m == 1)
					flags_torreta |= FLAG_TRIGGER_BUTTON;
				if (m == 2)
					flags_torreta |= FLAG_JOYSTICK_UP;
				if (m == 3)
					flags_torreta |= FLAG_JOYSTICK_DOWN;
				if (m == 4)
					flags_torreta |= FLAG_JOYSTICK_LEFT;
				if (m == 5)
					flags_torreta |= FLAG_JOYSTICK_RIGHT;
			}
			if (m != 0)
				c++;
		}
		key = 1;
		delay(500);
	}
}

PI_THREAD (thread_explora_teclado_PC) {
	int teclaPulsada;

	while (1) {
		delay(10); // Wiring Pi function: pauses program execution for at least 10 ms
```

```
		piLock(STD_IO_BUFFER_KEY);

		if (kbhit()) {
			teclaPulsada = kbread();

			switch (teclaPulsada) {

			case 's': //Start Impacto

				flags_player |= FLAG_START_DISPARO;

				printf("Tecla S pulsada!\n");
				fflush(stdout);
				break;

			case 'q':
				exit(0);
				break;

			default:
				printf("INVALID KEY!!!\n");
				break;
			}
		}

		piUnlock(STD_IO_BUFFER_KEY);
	}
}

// wait until next_activation (absolute time)
void delay_until(unsigned int next) {
	unsigned int now = millis();
	if (next > now) {
		delay(next - now);
	}
}
//Hebra Principal
int main() {
	fd =   serialOpen(   "/dev/serial/by-id/usb-Arduino__www.arduino.cc__0042_85734323313519170D1-if00",115200);

	delay(1000);
	TipoSistema sistema;

	unsigned int next;
```

```
// Configuracion e Inicializacion del sistema
ConfiguraSistema(&sistema);
InicializaSistema(&sistema);

//PLAYER
fsm_trans_t reproductor[] =
             { { WAIT_START, CompruebaStartDisparo, WAIT_NEXT,
                          InicializaPlayDisparo }, { WAIT_START,
                          CompruebaStartImpacto, WAIT_NEXT, InicializaPlayImpacto }, {
                          WAIT_NEXT, CompruebaStartImpacto, WAIT_NEXT,
                          InicializaPlayImpacto }, { WAIT_NEXT, CompruebaNotaTimeout,
                          WAIT_END, ActualizaPlayer }, { WAIT_END,
                          CompruebaFinalEfecto, WAIT_START, FinalEfecto }, { WAIT_END,
                          CompruebaNuevaNota, WAIT_NEXT, ComienzaNuevaNota }, { -1,
             NULL, -1, NULL }, };

fsm_t* player_fsm = fsm_new(WAIT_START, reproductor, &(sistema.player));

//TECLADO
teclado = &sistema.teclado;

fsm_trans_t columns[] = { { KEY_COL_1, CompruebaColumnTimeout, KEY_COL_2,
             col_2 }, { KEY_COL_2, CompruebaColumnTimeout, KEY_COL_3, col_3 }, {
             KEY_COL_3, CompruebaColumnTimeout, KEY_COL_4, col_4 }, { KEY_COL_4,
             CompruebaColumnTimeout, KEY_COL_1, col_1 }, { -1,
NULL, -1, NULL }, };

fsm_trans_t keypad[] = { { KEY_WAITING, key_pressed, KEY_WAITING,
             process_key }, { -1, NULL, -1, NULL }, };

fsm_t* columns_fsm = fsm_new(KEY_COL_1, columns, &sistema.teclado);
fsm_t* keypad_fsm = fsm_new(KEY_WAITING, keypad, &sistema.teclado);

//TORRETA
fsm_trans_t torreta[] =
             { { WAIT_START, CompruebaComienzo, WAIT_MOVE, ComienzaSistema }, {
                          WAIT_MOVE, CompruebaJoystickUp, WAIT_MOVE,
                          MueveTorretaArriba }, { WAIT_MOVE, CompruebaJoystickLeft,
                          WAIT_MOVE, MueveTorretaIzquierda }, { WAIT_MOVE,
                          CompruebaJoystickRight, WAIT_MOVE, MueveTorretaDerecha }, {
                          WAIT_MOVE, CompruebaJoystickDown, WAIT_MOVE,
                          MueveTorretaAbajo }, { WAIT_MOVE, CompruebaTriggerButton,
                          TRIGGER_BUTTON, DisparoIR }, { TRIGGER_BUTTON,
```

```
                                    CompruebaTimeoutDisparo, WAIT_MOVE, FinalDisparoIR }, {
                                    TRIGGER_BUTTON, CompruebaImpacto, WAIT_MOVE,
                                    ImpactoDetectado }, { WAIT_MOVE, CompruebaFinalJuego,
                                    WAIT_END, FinalizaJuego }, { -1, NULL, -1, NULL }, };

        fsm_t* torreta_fsm = fsm_new(WAIT_START, torreta, &sistema.torreta);

        next = millis();

        //Maquinas de Estado
        while (1) {
                fsm_fire(player_fsm);
                fsm_fire(columns_fsm);
                fsm_fire(keypad_fsm);
                fsm_fire(torreta_fsm);
                next += CLK_MS;
                delay_until(next);
        }
        return 0;
}
```