INGENIERÍA TÉCNICA INDUSTRIAL:

ELECTRÓNICA INDUSTRIAL

PROYECTO FIN DE CARRERA:

CONTROL DE PANTALLAS uOLED POR PUERTO SERIE USANDO UN MICROCONTROLADOR ARDUINO ATMEGA2560 PARA REPRESENTAR ANIMACIONES

AUTOR: LORENA GÓMEZ FERNÁNDEZ

TUTORES: RAMÓN BARBER CASTAÑO

DAVID GARCÍA GODOY

JULIO 2012

Agradecimientos:

Quiero empezar agradeciendo a mis padres su apoyo y motivación, no solo en este proyecto si no a lo largo de toda la carrera.

Menciono también a mi novio y a mis amigos, por estar siempre ahí a lo largo del desarrollo de este proyecto.

Por último me gustaría agradecer a mis dos tutores, a D. Ramón Barber y a D. David García, por su infinita ayuda y paciencia. Sin ellos no podría haber llevado a cabo este proyecto.

Contenido

Capítulo 1: INTRODUCCIÓN

1.1. MOTIVACIÓN

La robótica personal y de servicios requiere cada día de medios de interacción entre los humanos y los robots. En este ámbito, pensando en robots concebidos para interaccionar de forma personal y directa con el usuario, como son los robots de compañía o los robots lúdicos, se hace cada día más necesario el que el robot sea capaz de expresar su estado de una forma amigable. Así, en la robótica se ha intentado emular el modo en que lo hacen los humanos [1][2] de diferentes formas. Dentro de las formas de mostrar expresiones, aparecen soluciones basadas en imitar los ojos humanos [3]. En este sentido, en el roboticslab de la UC3M se estudian nuevas formas de expresión de robots, siendo una de ellas a través de displays que muestran imágenes o animaciones.

Dentro de este tipo de expresión se encuadran los desarrollos de este proyecto, en los que se desarrollan trabajos destinados a la expresión del robot a través de sus ojos, los cuales van a ser interpretados por los displays. Las expresiones se han decidido llevar a cabo a través de la reproducción de animaciones, que se elegirán previamente.

En el desarrollo de este proyecto, para gestionar los displays, se emplea un microcontrolador Arduino, que aparece como uno de los micros más usados por su sencillez, su facilidad de programación, el número de librerías disponibles, y su bajo coste. Además, debido a su pequeño tamaño, también puede ir incorporado en el robot cuyas expresiones se vayan a controlar.

Parte de la potencia y sencillez la proporciona el uso de protocolos serie, por lo que formarán parte del proyecto, algunos ya implementados y otros nuevos que se crearán.

1.2. OBJETIVOS

En este proyecto se busca la implementación de un sistema que permita la expresión de un robot personal o de servicios mediante el uso de displays, utilizando comunicación serie a través de un micro. El objetivo principal definido para el proyecto es:

- Controlar dos pantallas uOLED-128-G1 por puerto serie, utilizando un microcontrolador Arduino ATmega2560.

Este objetivo puede desglosarse en los siguientes objetivos de trabajo:

- Estudio de diferentes propuestas de displays y protocolos serie entre displays y micro.
- Desarrollo de un protocolo serie que permita la comunicación entre una aplicación y los displays.
- Encapsular las funciones de dicho protocolo en unas librerías.
- Elaboración de un primer cliente para probar la librería y el protocolo serie desarrollado.

Para ello ha sido necesario mirar propuestas de displays y de microcontroladores Arduino disponibles. También se ha realizado un estudio de Arduino y de su entorno de programación, así como del funcionamiento de sus pines y puertos serie y el encapsulado en librerías de su código.

En lo respectivo a los displays, se ha estudiado el funcionamiento de los finalmente elegidos y su protocolo serie, su conexión con la placa Arduino y sus librerías disponibles, llevando a cabo ejemplos para probar las funciones.

Además también ha sido necesario la implementación de un protocolo serie que recibe el microcontrolador Arduino y que realiza un multiplexado hacia las dos pantallas.

Para introducir dicho protocolo serie ha sido necesario la creación de una aplicación cliente en Linux, que envía los comandos hacia Arduino por puerto serie.

Las especificaciones de partida para el sistema que se pretende desarrollar son las siguientes:

- Los datos (animación que se quiere ejecutar, repeticiones, pantalla o pantallas a las que van dirigidas, etc.) se introducen a través de la aplicación cliente con un protocolo serie creado específicamente para este proyecto. Este protocolo llega a Arduino a través del puerto serie 0 (P.S.0), que es leído continuamente. Una vez que Arduino ha recibido la trama entera del protocolo se procesan los datos, comprobando que sean correctos y que constan de principio y final de trama. En la Figura 1 se observa el esquema de esta recepción de datos.

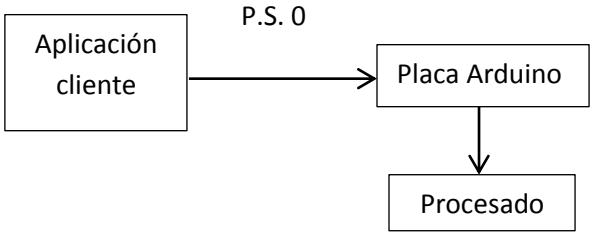

Figura 1: Esquema de recepción de datos

- Tras esto, los datos correspondientes para la ejecución de la animación requerida se envían siguiendo el protocolo serie propio de las pantallas por los puertos serie 1 y 2 (P.S.1 y P.S.2), es decir, Arduino hace el trabajo de multiplexor, tal como se muestra en la Figura 2.

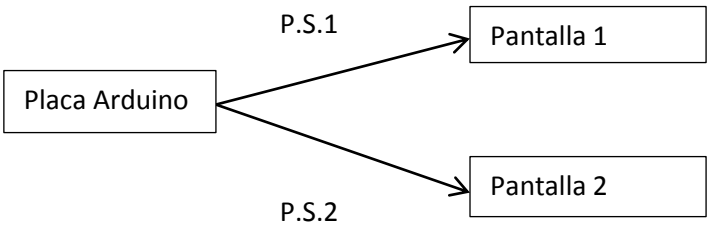

Figura 2: Esquema de envío de datos

- De esta manera, la comunicación entre la aplicación cliente y las pantallas se realiza de la siguiente manera:

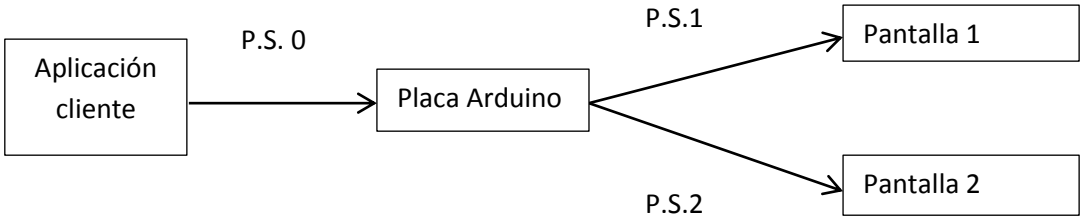

Figura 3: Esquema de comunicación entre aplicación y pantallas

- Una vez ejecutada la animación, las pantallas envían por puerto serie la respuesta. Si se ha ejecutado correctamente se recibe el *acknowledgement* (ack), y en caso contrario se recibe el *negative acknowledgement* (nack). (Ver Figura 4).

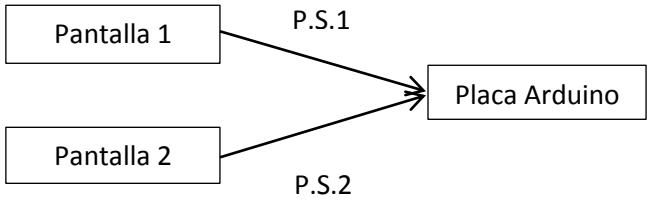

Figura 4: Esquema de recepción de datos de las pantallas

- La comunicación entre la aplicación cliente y Arduino también es reversible, como se observa en la Figura 5. En el caso en el que la trama recibida no sea correcta, Arduino envía un mensaje a la aplicación mediante el puerto serie 0 que esta debe mostrar.

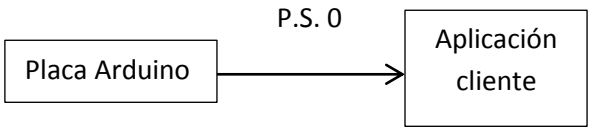

Figura 5: Esquema de comunicación entre Arduino y cliente

1.3. PARTES DEL DOCUMENTO

En el **capítulo 1** se describen las motivaciones y objetivos del proyecto, un resumen del por qué y el qué se va hacer.

En los **capítulos 2 y 3** se describen la placa Arduino y las pantallas elegidas respectivamente, así como sus características, conexiones, entorno de programación y demás datos necesarios para la comprensión del proyecto.

En el **capítulo 4** se encuentra la definición del protocolo serie que se sigue a la hora de mandar los comandos por el puerto serie 0 hacia el micro.

En el **capítulo 5** se explica la implementación llevada a cabo para la funcionalidad del proyecto: funciones empleadas, librerías y flujogramas.

Una vez explicado todo el proyecto y de ver cómo se ha desarrollado, en el **capítulo 6** se encuentran los resultados experimentales obtenidos, utilizando fotos como apoyo visual.

Por último, en el **capítulo 7** se observan las conclusiones obtenidas y las posibles ampliaciones del proyecto.

Capítulo 2: ARDUINO

2.1. INTRODUCCIÓN

Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar.

Arduino puede tomar información del entorno a través de sus pines de entrada y puede afectar a aquello que le rodea controlando luces, motores y otros actuadores con los pines de salida. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino y el entorno de desarrollo Arduino. Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar el micro a un ordenador, pero tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software.

Tiene una página web [4] en la que se puede encontrar todo lo necesario para programar en Arduino, desde la opción de descarga del entorno de desarrollo con sus instrucciones de instalación hasta una pequeña guía de programación. También se pueden encontrar ejemplos de programas y librerías que se pueden descargar. Además, cuenta con un foro en el que puedes exponer tus dudas y actualmente se está ampliando al español [5].

En esta página también se pueden encontrar los distribuidores a través de los cuales comprar las placas Arduino o los componentes necesarios para ser hechas a mano. Existe una gran variedad de placas Arduino ya montadas de fábrica, que depende del tamaño, número de pines o microcontrolador incorporado. De un mismo modelo existen varias versiones, ya que se van actualizando haciendo mejoras. Esta variedad de placas se puede ver en la Figura 6.

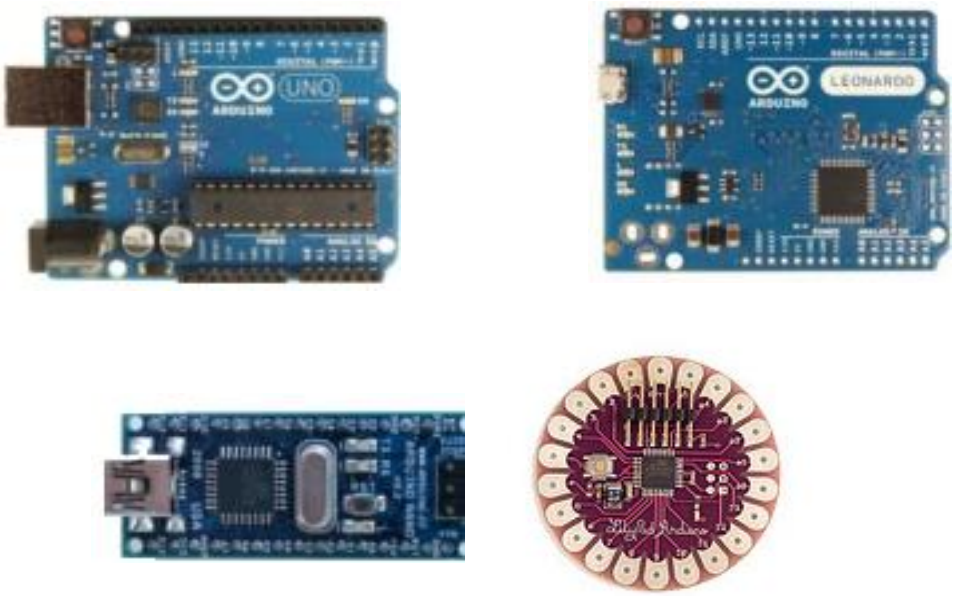

Figura 6: Distintos tipos de microcontroladores Arduino (Uno, Leonardo, Nano, LilyPad)

Para obtener información de las placas más nuevas y actualizadas es necesario visitar la web en inglés [6], puesto que las hojas de características no están traducidas aún.

A parte de las placas con microcontroladores, en la web de Arduino también podemos encontrar ampliaciones para dichas placas y que aumentan sus funcionalidades. Un ejemplo puede ser la opción de realizar conexiones libres de cables, dotar al microcontrolador de un zócalo para tarjetas de memoria SD o la posibilidad de que tenga conexión a internet. Estas placas shields son de la forma mostrada en la Figura 7:

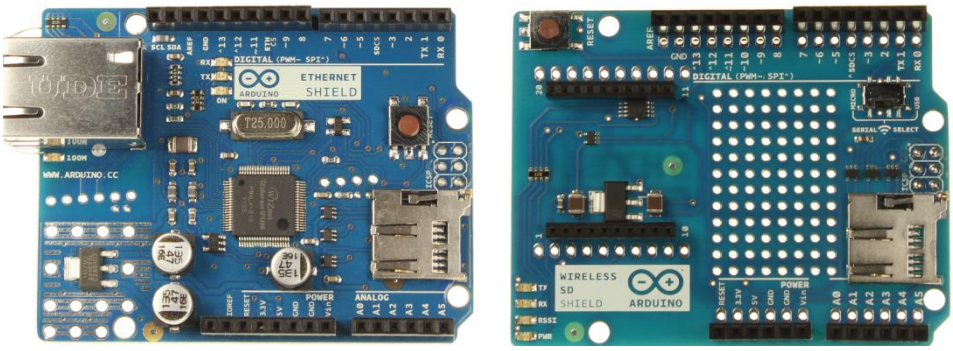

Figura 7: Placas shield de Arduino

2.2. PLACA ARDUINO MEGA 2560

2.2.1. INTRODUCCIÓN

Para este proyecto se ha elegido la placa Arduino Mega 2560 (ver Figura 8). Dentro de la diversidad existente, es la placa más grande, con el mayor número de pines digitales y analógicos y además tiene la opción de generar señales PWM. Estas características, aunque no se usen en este proyecto, se tienen en cuenta para futuras ampliaciones. La característica fundamental por la que se eligió esta placa es el número de puertos serie disponibles, con un total de 4, de los cuales se van a utilizar 3.

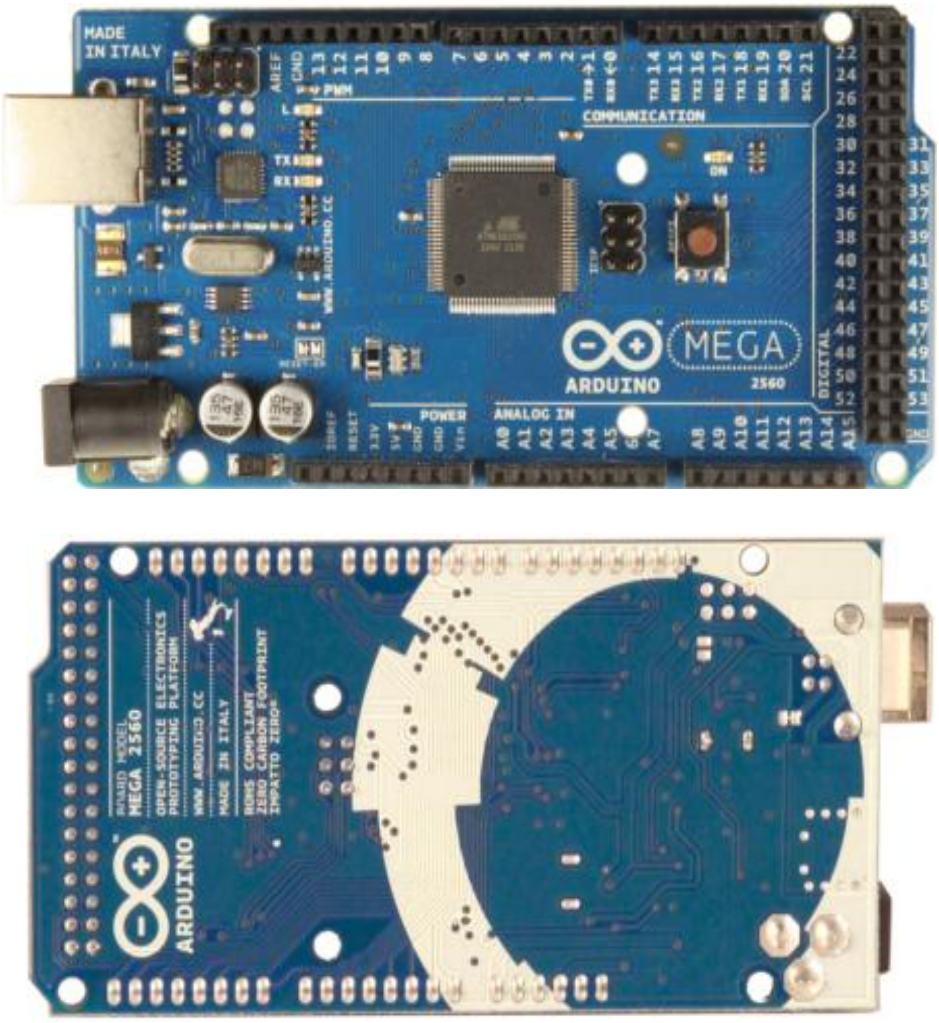

Figura 8: Arduino Mega 2560

Además, consta de varios pines de alimentación de 5V y de GND, por lo que admite varias conexiones sin necesidad de una placa de entrenamiento adicional. Esto va a ser necesario puesto que se deben conectar dos pantallas a la placa Arduino, y cada una de ellas necesita un pin de alimentación y otro de tierra.

Otra característica destacable es su pin digital 13, que va conectado a un LED que se enciende cuando el pin se encuentra en nivel alto. Esto es útil a la hora de familiarizarse con Arduino y su programación, puesto que sin llegar a la comunicación serie, se puede tener información de si se ha ejecutado el programa y si ha sido de forma correcta.

A parte de esto, la placa Arduino Mega 2560 puede ser alimentada a través de un ordenador o bien usando una fuente externa. Si se elige la alimentación por ordenador, esta conexión también une por el puerto serie 0 el ordenador y la placa Arduino. De elegirse una alimentación externa, el puerto serie 0 puede ser utilizado a través de sus pines correspondientes (TX0 y RX0) al igual que el resto de los puertos serie disponibles (4 en total).

La placa puede ser reseteada tanto físicamente, usando el botón disponible, o en la programación. Cada vez que es conectada a un ordenador también se resetea.

A pesar de la gran cantidad de pines, el Arduino Mega 2560 tiene unas dimensiones compatibles con la finalidad del proyecto, que será su incorporación a un robot personal o de servicios. Estas dimensiones son de 10,16 cm x 5,33 cm.

2.2.2. RESUMEN DE CARACTERÍSTICAS

En la Tabla 1 se muestra un resumen de las características de la placa Arduino Mega 2560. Algunas de ellas serán detalladas más adelante.

Microcontrolador	ATmega2560
Voltaje de funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (límite)	6-20V
Pines E/S digitales	54 (15 proporcionan salida PWM)
Pines de entrada analógica	16
Intensidad por pin	40mA
Intensidad en pin 3.3V	50mA
Memoria Flash	256 KB de las cuales 8 KB las usa el gestor de arranque(bootloader)
SRAM	8 KB
EEPROM	4KB
Velocidad de reloj	16 MHz
Puertos serie	4

Tabla 1: resumen del Arduino Mega 2560

2.2.3. ALIMENTACIÓN DE LA PLACA

El Arduino Mega puede ser alimentado a través de la conexión USB o con una fuente de alimentación externa (ver Figura 9). Si es por USB además se estaría utilizando el puerto serie 0. El origen de la alimentación se selecciona automáticamente.

Figura 9: Alimentación Arduino

Las fuentes de alimentación externas pueden ser o un transformador o una batería. El transformador se puede conectar usando un conector macho de 2.1mm con centro positivo en el conector hembra de la placa. Los cables de la batería pueden conectarse a los pines GND y V_{IN} en los conectores de alimentación.

La placa puede trabajar con una alimentación externa de entre 6 a 20 voltios. En el caso de que se sobrepase la intensidad permitida, el Arduino Mega 2560 cuenta con un sistema de protección, que corta la alimentación.

Con un voltaje inferior a 7V el pin de 5V puede proporcionar menos de 5 Voltios y la placa puede volverse inestable. Por otro lado si se usan más de 12V los reguladores de voltaje se pueden sobrecalentar y dañar la placa. Así pues, el rango recomendado es de 7 a 12 voltios.

Los pines de alimentación son los siguientes (ver Figura 10):

VIN. Es la entrada de voltaje a la placa Arduino cuando se está usando una fuente externa de alimentación. Se puede proporcionar voltaje a través de este pin, o, si se está alimentado a través de la conexión de 2.1mm, acceder a ella a través de este pin.

5V. Es la fuente de voltaje estabilizado usado para alimentar el microcontrolador y otros componentes de la placa. Esta puede provenir de V_{IN} a través de un regulador integrado en la placa, o proporcionada directamente por el USB u otra fuente estabilizada de 5V. Hay un total de tres pines de 5V, uno en la zona de alimentación y otros dos en la zona de pines digitales.

3V3. Es una fuente de voltaje a 3,3 voltios generada en el chip FTDI integrado en la placa. La corriente máxima soportada es de 50mA.

GND. Pines de toma de tierra. Tiene un total de 5: dos en la zona de alimentación, dos en la zona de los pines digitales y otro más junto a los pines de PWM.

Figura 10: Pines de alimentación

2.2.4. PINES DE LA PLACA

Además de los pines de alimentación descritos anteriormente, podemos encontrar otros pines como:

- PINES DIGITALES

Cada uno de los 54 pines digitales (ver Figura 11) pueden utilizarse como entradas o salidas usando las funciones *pinMode()*, *digitalWrite()*, y *digitalRead()*. Operan a 5 voltios. Cada pin puede proporcionar o recibir una intensidad máxima de 40mA y tiene una resistencia interna (desconectada por defecto) de 20-50kOhms. En el mismo bloque de pines, podemos encontrar arriba dos pines de 5V, y abajo otros dos pines de GND.

Figura 11: Pines digitales

- PINES PUERTO SERIE

Los pines del conexionado serie son: Serie 0: 0 (RX) y 1 (TX); Serie 1: 19 (RX) y 18 (TX); Serie 2: 17 (RX) y 16 (TX); Serie 3: 15 (RX) y 14 (TX). Usado para recibir (RX) y transmitir (TX) datos a través de puerto serie TTL. Los pines Serie 0 (RX) y 1 (TX) están conectados a los pines correspondientes del chip FTDI USB-a-TTL, es decir, que están conectados a la conexión a través de USB con el ordenador (ver Figura 12).

Figura 12: Pines del conexionado serie

- PINES INTERRUPCIONES

Se pueden encontrar distribuidos por los pines de la placa Arduino. Las interrupciones externas se corresponden con los pines: 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3), y 21 (interrupción 2). Estos pines se pueden configurar para lanzar una interrupción en un valor LOW (0V), en flancos de subida o bajada (cambio de LOW a HIGH (5V) o viceversa), o en cambios de valor.

- PINES ANALÓGICOS

El Mega tiene 16 entradas analógicas (ver Figura 13), y cada una de ellas proporciona una resolución de 10bits (1024 valores). Por defecto se mide de tierra a 5 voltios, aunque es posible cambiar la cota superior de este rango usando el pin AREF y la función *analogReference()*. El nombre de los pines es alfanumérico, todos van precedidos de 'A'.

Figura 13: Pines analógicos

- PINES DE PWM

Estos pines (ver Figura 14) proporcionan una salida PWM (Pulse Wave Modulation, modulación de onda por pulsos) de 8 bits de resolución (valores de 0 a 255) a través de la función *analogWrite()*.

Figura 14: Pines para generar señal PWM

- OTROS PINES EN LA PLACA:

AREF. Voltaje de referencia para las entradas analógicas. Usado por *analogReference()*. Ver Figura 15.

Figura 15: Pin AREF

Reset. Suministra un valor de 0V para reiniciar el microcontrolador. Típicamente es usado para añadir un botón de reseteo a los shields que no dejan acceso a este botón en la placa. Ver Figura 16.

Figura 16: Pin reset

LED 13. Hay un LED integrado en la placa conectado al pin digital 13. Cuando este pin tiene un valor HIGH (5V) el LED se enciende y cuando este tiene un valor LOW (0V) este se apaga. Como se aprecia en la Figura 17, se encuentra junto a los leds de comunicación serie, que se encienden dependiendo de si transmite (TX) o recive (RX) por cualquiera de los puertos serie.

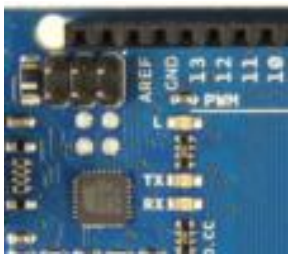

Figura 17: Led 13

2.3. ENTORNO DE DESARROLLO PARA ARDUINO

El entorno de desarrollo para Arduino se encuentra disponible de forma gratuita en la página web de Arduino [7]. Se puede elegir la versión dependiendo del sistema operativo utilizado. Las versiones se van actualizando cada poco tiempo, y si existe una actualización disponible salta un aviso cada vez que se abre el entorno. Para este proyecto se ha utilizado la versión 22.

El entorno (ver Figura 18) está constituido por un editor de texto para escribir el código, un área de mensajes, una consola de texto, una barra de herramientas con botones para las funciones comunes, y una serie de menús. Permite la conexión con el hardware de Arduino para cargar los programas y comunicarse con ellos.

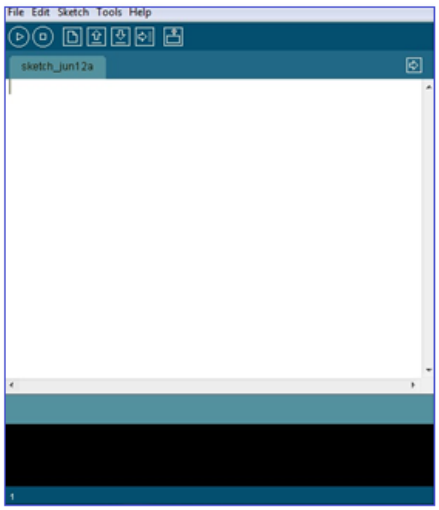

Figura 18: Entorno Arduino

Para conectar la placa con el entorno de desarrollo, se necesita seleccionar el tipo de placa, como se ve en la Figura 19.

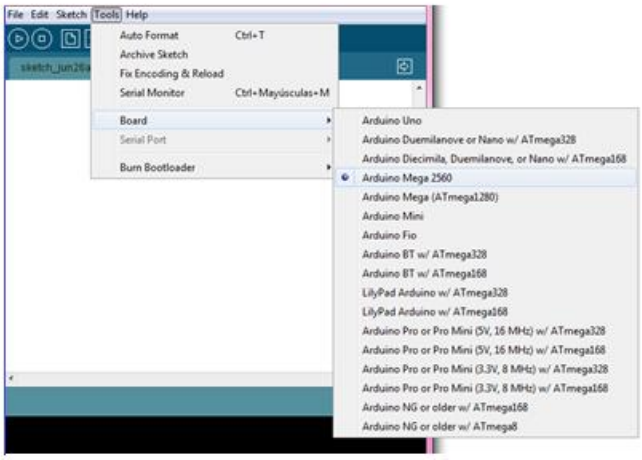

Figura 19: Selección de placa

Una vez seleccionada la placa es necesario seleccionar el puerto serie en el que se encuentra, para poder comenzar la comunicación. El número del puerto serie cambia si se encuentra en Linux o en Windows. En este caso en Windows se encuentra en el **COM3**, mientras que en Linux, pese a poder cambiar, suele encontrarse en el **/dev/ttyACM0.** En la Figura 20 y en la Figura 21 se muestran la selección del puerto serie.

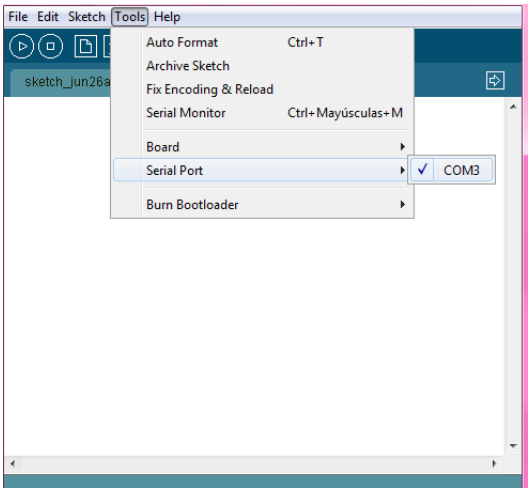

Figura 20: Selección del puerto en Windows

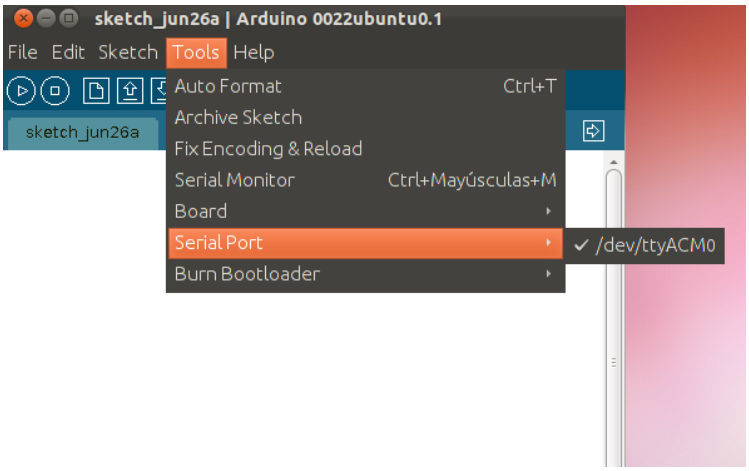

Figura 21: Selección del puerto en Linux

Arduino utiliza para escribir el software lo que denomina "sketch" (programa). Estos programas son escritos en el editor de texto. En el área de mensajes se muestra información mientras se cargan los programas y también muestra los errores ocurridos al compilar o cargar el programa, o si estos procesos se han realizado satisfactoriamente. La consola muestra el texto de salida para el entorno de Arduino incluyendo los mensajes de error completos y otras informaciones. La barra de herramientas (ver Figura 22) permite verificar el proceso de carga (compilar), pararlo, creación, apertura y guardado de programas, descargar el programa en la placa y la monitorización serie.

Figura 22: Barra de herramientas

Para empezar a programar en Arduino, a parte de la información encontrada en su página web [4], se pueden consultar manuales escritos para tal efecto [8].

2.3.1. LIBRERÍAS DE ARDUINO

El entorno de programación de Arduino cuenta con una gran variedad de librerías. Para poder usarlas simplemente se deben añadir al principio del programa, tal como se muestra en la Figura 23.

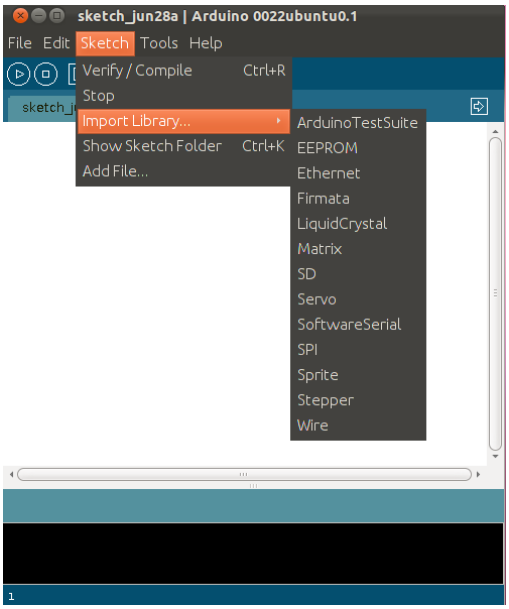

Figura 23: Selección de librerías

En la web de Arduino existen más librerías que pueden ser descargadas gratuitamente [9]. Una vez que se han descargado, deben introducirse en el entorno de programación, en la carpeta destinada para tal efecto, además de introducirlas en el hardware de Arduino. Todo este proceso se encuentra explicado en el apartado de librerías de la página web de Arduino anteriormente mencionada.

2.4. MONITORIZACIÓN SERIE

Arduino posee su propia monitorización serie a través de la cual escribir o recibir datos por puerto serie (ver Figura 24).

Muestra los datos enviados desde la placa Arduino (placa USB o serie). Para enviar datos a la placa, se teclea el texto y se pulsa el botón "send" o enter, que no forma parte de la trama. Es necesario seleccionar la velocidad (baud rate) en el menú desplegable y que a su vez coincida con la configurada en el *Serial.begin()* dentro del programa. En este proyecto se trabajará a una velocidad de 9600 baudios.

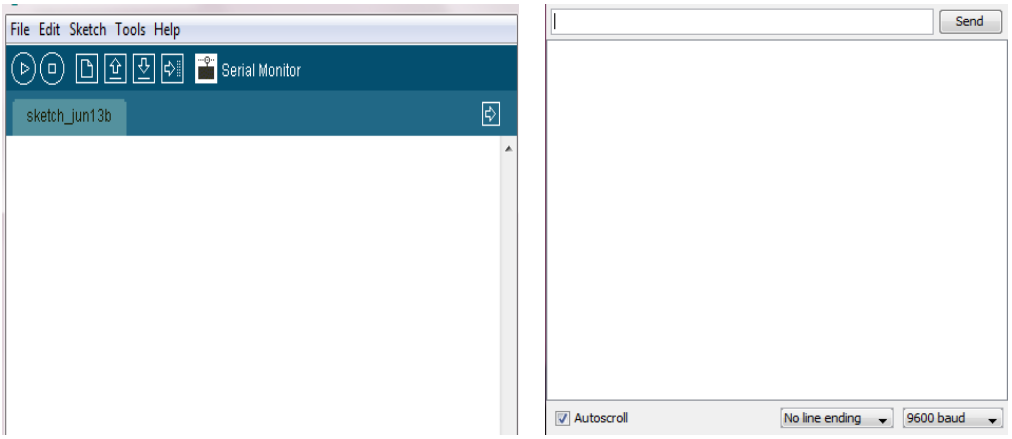

Figura 24: Monitor serie

Cualquier mensaje que se mande imprimir en el programa (usando la función *Serial.print()* o *Serial.println()*) se muestra en este monitor serie. También realiza la función contraria: cualquier comando que se escriba se manda por puerto serie esperando a ser leído.

El puerto serie por el que lee o envía es el puerto serie 0, ya que si se puede acceder a este monitor serie es que el Arduino está conectado al ordenador a través del USB, y como se ha mencionado anteriormente, esta conexión está ligada al puerto serie 0.

Para que la introducción de los comandos sea más fácil, se ha creado una aplicación cliente, donde se recogen varias opciones de comandos. Esto se ha realizado para que, habiendo elegido unas tramas predeterminadas, no haya error al mandarlas a Arduino, puesto que se eligen pulsando simplemente una tecla. Estas tramas se muestran en la Tabla 2, donde quedan recogidos los comandos enviados, el comando que se debe introducir en la aplicación y su funcionalidad.

Número de comando	Trama mandada	Qué hace
1	DP00023	Parpadear
2	DP00101	Sonreír
3	DP10703	Sospechar (derecha)
4	DP20321	Sorprenderse (izquierda)
5	DP00504	Permanecer triste
6	DP00702	Sospechar
7	DP00321	Sorprenderse
8	DP00104	Permanecer sonriendo
9	DP00205	Dejar de sonreír

Tabla 2: Comandos predefinidos

En el caso de que se quiera mandar un comando fuera de las opciones, se puede introducir a través de un hyperterminal, si en su configuración se liga al puerto serie correspondiente.

Capítulo 3: DISPLAYS

3.1. INTRODUCCIÓN

Para implementar las animaciones que se tienen que ejecutar se necesitan displays o pantallas que cumplan los siguientes requisitos: reproducir animaciones, ser conectadas a través de puerto serie, fácil manejo, con librerías implementadas y que dispongan de un medio para guardar los gifs, como puede ser una tarjeta de memoria SD. Además, como su finalidad está pensada para ir acoplada a un robot y que haga la función de ojos, las dimensiones apropiadas también es un requisito a cumplir.

En el mercado existen varias marcas que trabajan con modelos de pantallas que se pueden conectar a Arduino, como por ejemplo:

Iteadstudio. [10]. Entre algunos modelos podemos encontrar:

- Serial 2.2" TFT LCD Screen Module: ITDB02-2.2SP. Ver Figura 25.

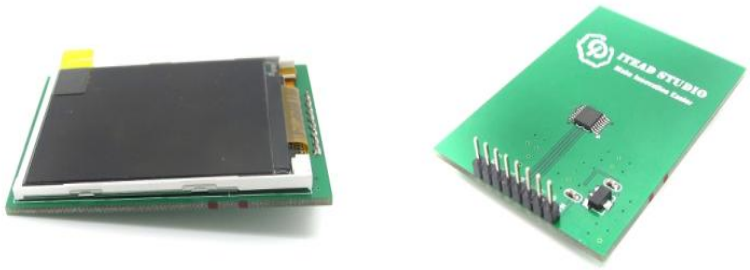

Figura 25: Pantalla ITDB02-2.2SP

Se trata de una pantalla LCD capaz de reproducir animaciones y se manejan fácilmente por puerto serie, pero no disponen de almacenaje de imágenes, por lo que habría que mandárselas desde el código. Sus hojas de características pueden ser descargadas de la web [11].

Cuenta con unas dimensiones de 43.0mm X 6.05mm X 1.6mm, medidas adecuadas a la finalidad que deben cumplir.

- 2.4" TFT LCD Screen Module: ITDB02-2.4E

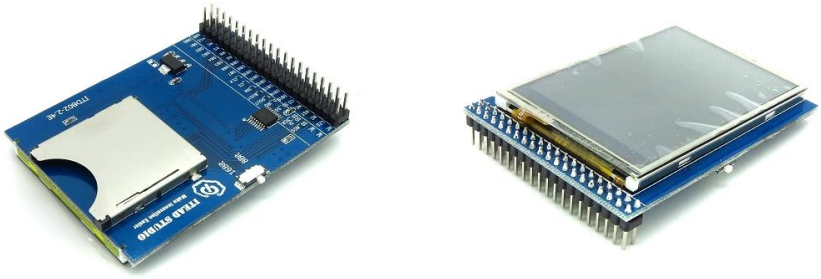

Figura 26: Pantalla ITDB02-2.4E

Como se aprecia en la Figura 26, sí dispone de zócalo para tarjetas SD, además es táctil, dándole más funcionalidad a las pantallas. No tiene conexión a través de puerto serie, por lo que su adaptación a Arduino es muy complicada. Se le puede añadir una placa shield que se acopla a la placa Arduino Mega. Dispone de librería implementada. Sus características se pueden observar y descargar en la web [7].

Tiene unas dimensiones de 65.53mm X 50.8mm X 1.6mm, por lo que serían un medidas demasiado grandes para que formen parte de este proyecto.

4Dsystems.

Trabaja con smart displays, es decir, pantallas inteligentes, puesto que se conectan a través de puerto serie. Constan de zócalo para tarjetas de memoria SD, por lo que no hace falta que se le envíen las imágenes, simplemente la dirección de memoria en la que se encuentran. Un ejemplo de estas pantallas se muestra en la Figura 27.

Figura 27: Pantalla 4D systems

En su página web [12] se pueden encontrar todos sus productos disponibles, así como sus descripciones y hojas de características. A parte de pantallas también tiene disponibles placas shield, procesadores o placas de pruebas, como se muestra en la Figura 28.

Figura 28: Productos 4D systems

Existe mucha información en la web sobre estas pantallas, y varias librerías implementadas para su uso. Su protocolo serie es bastante sencillo. Existen varios modelos con distintas dimensiones, así que se puede elegir el que más se acerque a las especificaciones de cada proyecto.

Por estas características, son las pantallas elegidas para realizar este proyecto, puesto que cumplen con todos los requisitos.

3.2. DISPLAYS ADAPTADOS A ARDUINO

Existen unas placas shield para Arduino que permiten utilizar sus pantallas inteligentes OLED directamente con Arduino. Una placa shield es aquella que se puede acoplar al Arduino sin que ésta pierda sus pines (ver Figura 29).

Figura 29: Arduino + placa shield

Como en este proyecto las pantallas irán acopladas a un robot, no se necesita la placa shield, pero sí la característica de los displays de poder hacer una conexión serie con Arduino (ver Figura 30).

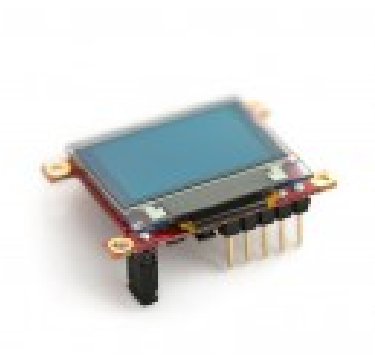

Figura 30: Pantalla sin placa shield

Estas pantallas permiten mostrar gráficos mediante un sencillo protocolo serie, pudiéndose dibujas cosas simples como píxeles, texto, rectángulos y círculos. Además disponen de un zócalo para tarjetas de memoria micro SD por lo que pueden almacenar y reproducir imágenes, vídeos o animaciones. Esta última característica será la utilizada en el proyecto.

Se alimentan con 5V y su consumo en general es muy bajo. Por su tecnología OLED, se ven extremadamente bien y son ideales para utilizar incluso a plena luz del sol (ver Figura 31).

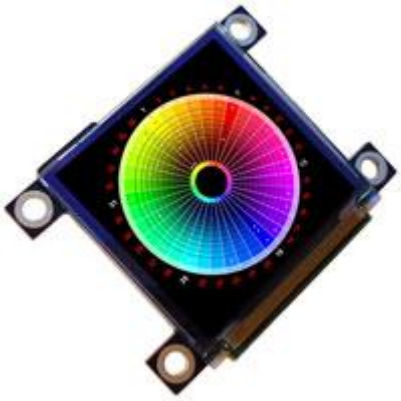

Figura 31: Resolución de la pantalla

Dentro del módulo se encuentran potentes gráficos, texto, imágenes, animación y otras propiedades. Ofrece un simple pero efectivo interfaz serie para cualquier microcontrolador que se pueda comunicar por puerto serie.

Al permitir al usuario utilizar el entorno de programación que desee, estas pantallas ofrecen una de las soluciones gráficas integradas más flexibles disponibles. Existen además varios modelos con la misma base y lo único que cambia es el tamaño de la pantalla.

3.3. DISPLAY uOLED-128-G1

3.3.1. INTRODUCCIÓN

Dentro de la variedad existente, en este caso se ha elegido el modelo uOLED-128-G1 (ver Figura 32) cuyas dimensiones se ajustan a las necesidades del proyecto.

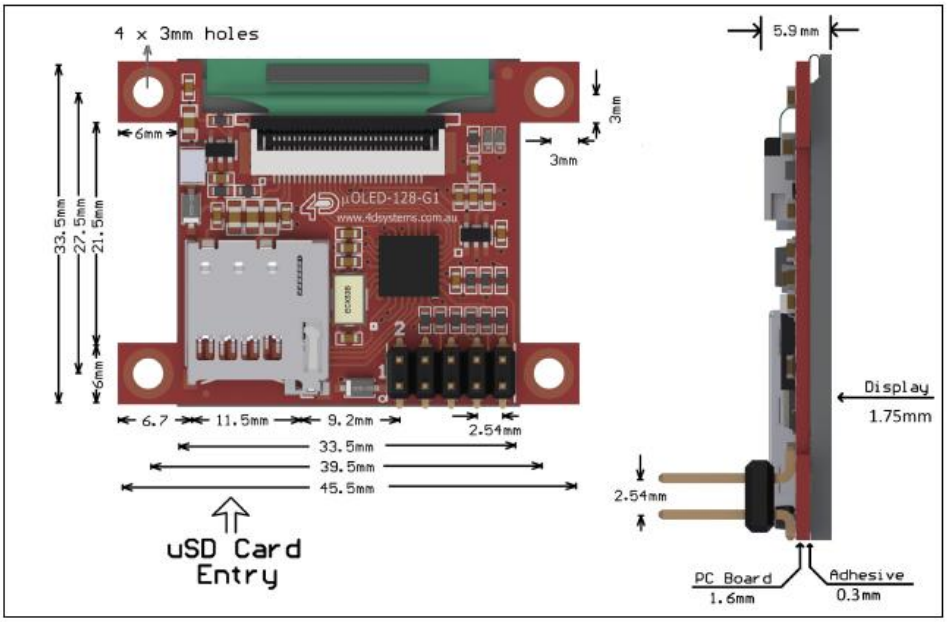

Figura 32: uOLED 128 G1.

Para aprender a manejarlas se ha utilizado una guía encontrada en la web [13], en la que explica cómo se conecta la placa y cómo se obtienen las imágenes, aunque se ha basado en el manual de instrucciones de las pantallas, que se puede descargar de la red [14], así como el manual con todos los protocolos serie necesarios para las pantallas [15].

3.3.1. PROPIEDADES

El uso de las pantallas uOLED es una solución económica para el usuario que quiera una interfaz gráfica. En concreto el modelo uOLED 128 G1 tiene una resolución de 128 x 128, con 65K colores. La pantalla es una pantalla PMOLED, con 1.5" de diagonal. Tiene cerca de 180° de ángulo de visión.

Las dimensiones totales son de 45.5x33.5x6.1mm, con un área total activa de 27mmx27mm.

En la parte de atrás se encuentran 5 pines fáciles de conectar con otro dispositivo: alimentación (VCC, GND), con un rango de operación entre 4.0V y 5.5V con una sola fuente; puerto serie (TX, RX), por lo que se puede manejar a través del puerto serie del Arduino; y un pin de RESET. Además cuenta con un pin dispuesto para el sonido.

Cuenta con el procesador 4D-Labs GOLDELOX-SGC, un controlador de gráficos serie inteligente diseñado para interactuar con diversos diplays OLED y LCD. Dentro del chip están integrados potentes gráficos.

Posee un zócalo para memoria micro SD, en la que se pueden cargar iconos, imágenes, animaciones o vídeos que más tarde se reproducirán en la pantalla. Admite tarjetas desde 64MB hasta 2GB.

A parte de reproducir imágenes, iconos y vídeos a todo color, usando un conjunto de funciones y algoritmos desarrollado en alto nivel se pueden dibujar líneas, círculos, texto y mucho más. Admite todos los caracteres y fuentes disponibles en Windows siempre que sean importadas como fuente externa.

3.3.2 APLICACIONES

Debido a sus propiedades, las aplicaciones de estas pantallas son variadas. Pueden ser utilizadas como solución general de gráficos integrados, puesto que se puede hacer reproducir una gran variedad de medios, como videos, animaciones, o texto.

Más concretamente, se podrían utilizar como sistemas de control en un ascensor, puesto que tienen el tamaño ideal. Además estarían bien valoradas para realizar un sistema de displays en el sector de la automoción.

En el sector industrial, serían útiles en el control industrial y en la robótica. Esto último es la aplicación desempleada en el proyecto, puesto que se ha hecho con vistas a que las pantallas vayan acopladas a un robot para realizar la función de ojos.

Como aplicaciones domésticas o de uso general, podríamos encontrar estas pantallas como parte de la automatización de una casa inteligente, ya que al poder unirse a microcontroladores puede mostrar en todo momento el estado de la casa. En este apartado también de debería considerar la opción de utilizarlas como parte de una instalación de juegos recreativos.

Habría que tener en cuenta que al ser de pequeño tamaño, podría tener aplicaciones médicas, o para sistemas de navegación GPS.

Por último, otra de sus posibles aplicaciones sería el control de sistemas de seguridad y acceso. Esto se podría aplicar a todos los ámbitos, ya sea sector industrial, de la medicina, o el doméstico.

3.4. CONEXIÓN DE LA PANTALLA AL ARDUINO

En la Tabla 3 se muestran los pines que posee la pantalla. Para mostrar imágenes se van a utilizar solamente los pines 1, 3, 5, 7 y 9.

Pin	Símbolo	E/S	Descripción
1	VCC	E	Principal fuente de voltaje. Protegido de polaridad reversible. El rango está entre 4.0 V y 5.5 V. Valor nominal de 5V
2	NC	-	No conectado
3	TX	S	Pin de transmisión serie asíncrona. Conectar este pin al correspondiente del microcontrolador que reciva la señal serie(RX). El microcontrolador recibe los datos del uOLED-128-G1(SGC) a través de este pin. Es tolerante a velores superiores a los 5 voltios.
4	SONIDO	S	Pin de salida de generación de sonido. Conecte este pin a un circuito simple de altavoz. Dejar abierto si no se usa.
5	RX	E	Pin de recepción serie asíncrona. . Conectar este pin al correspondiente del microcontrolador que transmita la señal serie(TX). El microcontrolador transmite comandos y datos al uOLED-128-G1(SGC) a través de est pin. Es tolerante a velores superiores a los 5 voltios.
6	SWITCH	E	Pin de entrada multi botón o joystick. Tiene la opción de conectar de 1 a 5 pulsadores. Si se conecta a GND al encenderse se manda un programa desde la tarjeta de memoria.
7	GND	P	Pin de tierra.
8	GND	P	Tierra. También proporciona una fácil conexión a tierra al pin 7.
9	RESET	E	Señal de reset maestra. Puede no ir conectada a ningún pin del micro.
10	3.3Vout	P	Salida regulada de 3.3V. disponible corriente superior a 50mA para suministrar circuitos externos.

Tabla 3: Pines del uOLED

En la Figura 33 se pueden observar las conexiones necesarias. El pin número 1 es el de la alimentación, por lo que se unirá a +5v (VCC); el pin número 3 es el TX y se une con el RX de Arduino; el número 5 es el RX, unido al TX de Arduino; por último, el pin 9 es el de reseteo que se uniría al pin designado como reseteo en el código. En este proyecto no se ha asignado ningún pin de reseteo, por lo que este pin quedaría sin conectar.

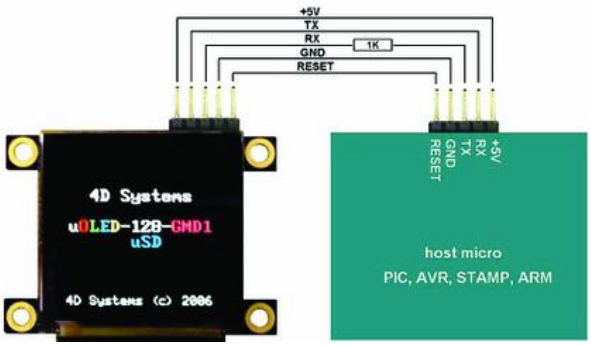

Figura 33: Conexión pantalla + microcontrolador

3.5. LIBRERÍA DE FUNCIONES

En la web hay disponibles varias librerías para utilizar con más facilidad estas pantallas, pero están concebidas para usar una sola, por lo que hubo que hacer modificaciones para tener en cuenta ambas pantallas y los distintos puertos serie. Se ha tomado como referencia una librería implementada por Óscar González que se puede descargar gratuitamente [14].

A continuación se muestra un listado de las funciones de la librería que se pueden usar directamente en el código:

- *char OLED_GetResponse(e_port puerto)*. Obtiene la respuesta de las pantallas. Se llama desde otras funciones.

- *void OLED_Init(e_port puerto)*. Inicializa las pantallas. Se llama desde el programa principal. Hay que indicar cuál de las dos pantallas se quiere inicializar.

- *int GetRGB(int red, int green, int blue, e_port puerto)*. Con ella se determina la intensidad de los colores básicos, el rojo el verde y el azul, a partir de los cuales se obtienen el resto de colores que se pueden utilizar en las demás funciones.

- *void OLED_Clear(e_port puerto)*. Limpia la pantalla, es decir, elimina cualquier imagen que podría haberse quedado reproducida en la pantalla.

- *void OLED_PutPixel(char x, char y, int color, e_port puerto)*. Dibuja un punto. Hay que indicarle las coordenadas, el color y la pantalla en la que se va a dibujar.

- *void OLED_DrawLine(char x1, char y1, char x2, char y2, int color, e_port puerto)*. Dibuja una línea. Hay que indicar las coordenadas de inicio y las del final de la recta, el color y la pantalla en la que se va a dibujar.

- *void OLED_DrawRectangle(char x, char y, char width, char height, char filled, int color, e_port puerto)*. Dibuja un rectángulo. Necesita saber las coordenadas de inicio, anchura, altura, si se va a rellenar, y el color, además de la pantalla ejecutora.

- *void OLED_DrawCircle(char x, char y, char radius, char filled, int color, e_port Puerto)*. Dibuja un círculo. Se necesita introducir las coordenadas del origen del círculo, el radio, si va a estar relleno, el color y la pantalla en la se va a dibujar el círculo.

- *void OLED_SetFontSize(char FontType)*. Elige la fuente con la que se va a escribir el texto. Admite la mayoría de las fuentes de Windows. No utiliza el puerto serie por lo que no es necesario especificar la pantalla.

- *void OLED_DrawText(char column, char row, char font_size, char *mytext, int color, e_port Puerto)*. Imprime un texto. Como argumentos, pide la columna y la fila en la que empezar a escribir, la fuente que debe utilizar, el texto que se debe escribir y el color que se va a usar. Por supuesto, también las pantalla a la que va dirigida.

- void OLED_DrawSingleChar(char column, char row, char font_size, char MyChar, int color). Imprime un solo carácter. Los argumentos son los mismos que los de la función *void OLED_DrawText()*, con la diferencia de que en vez de un texto únicamente se introduce un carácter.

En el código solamente se usarán las funciones *void OLED_Init(e_port puerto)* y *void OLED_Clear(e_port puerto).*

Para una primera toma de contacto con las pantallas y antes de trabajar con su protocolo serie propio, se usaron el resto de funciones para empezar a tener noción de la comunicación serie, puesto que se tuvieron que modificar todas para que actuaran a través del puerto serie 1 o del puerto serie 2.

Estas funciones no utilizan las animaciones, imágenes o vídeos que puedan estar en la tarjeta micro SD, si no que se usan para dibujar en la pantalla, ya sea una línea, círculo, rectángulo o escribir un texto.

3.6. VISUALIZACIÓN DE IMÁGENES DE LA SD

Las animaciones que se van a usar en este proyecto fueron realizadas en otro proyecto anterior, también en el roboticslab de la UC3M [16].

Teniendo las características necesarias, como son el número de imágenes y el tiempo entre imágenes, se puede usar cualquier animación, puesto que una vez que los tengamos se introducen en la tarjeta micro SD. Cuando las imágenes están en la tarjeta, se introduce en el zócalo (ver Figura 34) habilitado para tal acción.

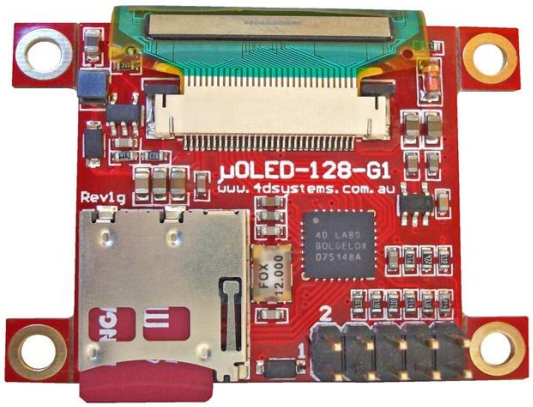

Figura 34: Zócalo para tarjetas

Una vez introducida, se llama a las imágenes desde el código. Para ello es necesario saber la dirección de memoria en la que se encuentra la imagen, vídeo o animación que queremos mostrar por pantalla. En este proyecto se quiere mostrar animaciones, por lo que los datos pasados por puerto serie serán, a parte de la dirección de memoria, la altura, ancho, bits de color, número de imágenes y retraso entre imágenes. Todos estos valores se introducen en forma hexadecimal. Se mandan a través del puerto serie uno a uno, siguiendo el protocolo serie de las pantallas, como se observa en la Figura 35.

```
void loop(){

  Serial1.print(0x40,BYTE);//@
  Serial1.print(0x56,BYTE);//v
  Serial1.print(0x00,BYTE);//x
  Serial1.print(0x00,BYTE);//y
  Serial1.print(0x80,BYTE);//width
  Serial1.print(0x80,BYTE);//heigh
  Serial1.print(0x10,BYTE);//colourmode
  Serial1.print(0xC8,BYTE);//delay
  Serial1.print(0x00,BYTE);//frames(msb)
  Serial1.print(0x06,BYTE);//frames(lsb)
  Serial1.print(0x00,BYTE);//sectorad(hi)
  Serial1.print(0x00,BYTE);//sectorad(mid)
  Serial1.print(0x00,BYTE);//sectorad(lo)
```

Figura 35: Código para mostrar una animación

3.6.1. PROTOCOLO SERIE PARA REPRODUCIR ANIMACIONES

Las pantallas uOLED-128-G1 cuentan con un protocolo propio, el cual ha de seguirse para poder darle una funcionalidad completa.

En concreto, en este proyecto se va a usar el protocolo para poder reproducir animaciones que se encuentren guardadas en las tarjetas de memoria que van a acopladas a las pantallas.

Los datos puestos a introducir para formar el protocolo se pueden encontrar en las hojas de propiedades de las animaciones. Como se ve en la Figura 36, en el apartado de Goldelox Data aparecen todos los datos siguiendo el orden del protocolo serie de las pantallas.

```
File "abriendo_mediob_derecho_lv6.gif" (abriendo_mediob_derecho_lv6.gif)
Sector Address 0x000180
X = 0 Y = 0 Width = 128 Height = 128 Bits = 16 Frames = 5 Delay = 255

Display Movie from Memory Card (Serial Command):
Syntax:
@, V, x, y, width, height, colourMode, delay, frames(msb), frames(lsb),
SectorAdd(hi), SectorAdd(mid), SectorAdd(lo)

Goldelox Data:
0x40, 0x56, 0x00, 0x00, 0x80, 0x80, 0x10, 0xFF, 0x00, 0x05, 0x00, 0x01, 0x80
Picaso Data:
0x40, 0x56, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x80, 0x10, 0xFF, 0x00,
0x05, 0x00, 0x01, 0x80

4DSL command:
Video(0, 0, 128, 128, 0x10, 255, 5, 0x000180)
```

Figura 36: Hoja de valores de la animación

En este caso únicamente se han mostrado las características de una animación, pero debe existir lo mismo para cada una de las animaciones que se quiera representar.

Según los manuales de las pantallas, los comandos del protocolo serie son los siguientes:

- **@** y **V (40hex y 56hex)**: comandos que identifican a este protocolo serie. Es el que indica que se van a mostrar unas animaciones incluidas dentro de la tarjeta de memoria.
- **x** e **y**: coordenadas que se usan de referencia para situar en la pantalla la animación.
- **Width** y **height**: ancho y altura de la animación. Estos valores pueden hacer que la animación se vea más pequeña o que abarque toda la pantalla.
- **colorMode**: se selecciona el modo del color.
- **Delay**: tiempo entre imagen e imagen que conforman la animación.
- **Frames**: es el número de imágenes de las que consta la animación. Se divide en 2 sectores, el medio (**frames(msb)**) y el bajo (**frames(lsb)**).
- **SectorAdd**: se trata de la dirección de memoria en la que se encuentra la animación. Se divide en 3 sectores, alto medio y bajo (**SectorAdd(hi)**, **SectorAdd(mid)**, **SectorAdd(lo)**).

En las hojas de características de las animaciones también se muestra el protocolo serie que se debe seguir, como se ve en la Figura 37:

```
Display Movie from Memory Card (Serial Command):
Syntax:
@, V, x, y, width, height, colourMode, delay, frames(msb), frames(lsb),
SectorAdd(hi), SectorAdd(mid), SectorAdd(lo)
```

Figura 37: Parámetros protocolo serie de las pantallas

Los valores de cada uno de los parámetros aparecen en el apartado Goldelox Data (ver Figura 38).

```
Goldelox Data:
0x40, 0x56, 0x00, 0x00, 0x80, 0x80, 0x10, 0xFF, 0x00, 0x05, 0x00, 0x01, 0x80
```

Figura 38: Valores de los parámetros

Estos datos, a la hora de implementar el sistema, se encapsularán en una librería que facilite cualquier modificación.

Todos los valores mencionados anteriormente se mandan uno a uno usando la función de Arduino *Serial.println()*. Esta función se modifica para especificar a través de qué puerto serie se va a enviar la información. En la Figura 39 se muestra un ejemplo implementado en el entorno de desarrollo de Arduino, en el que primero se manda por el puerto serie 1 y luego al puerto serie 2.

```
//abriendo_medio_izquierdo
Serial1.print(0x40,BYTE);
Serial1.print(0x56,BYTE);
Serial1.print(0x00,BYTE);
Serial1.print(0x00,BYTE);
Serial1.print(0x80,BYTE);
Serial1.print(0x80,BYTE);
Serial1.print(0x10,BYTE);
Serial1.print(0xC8,BYTE);
Serial1.print(0x00,BYTE);
Serial1.print(0x06,BYTE);
Serial1.print(0x00,BYTE);
Serial1.print(0x00,BYTE);
Serial1.print(0x00,BYTE);
 //abriendo_medio_dcho
Serial2.print(0x40,BYTE);
Serial2.print(0x56,BYTE);
Serial2.print(0x00,BYTE);
Serial2.print(0x00,BYTE);
Serial2.print(0x80,BYTE);
Serial2.print(0x80,BYTE);
Serial2.print(0x10,BYTE);
Serial2.print(0xC8,BYTE);
Serial2.print(0x00,BYTE);
Serial2.print(0x06,BYTE);
Serial2.print(0x00,BYTE);
Serial2.print(0x00,BYTE);
Serial2.print(0x00,BYTE);
```

Figura 39: Ejemplo protocolo serie puertos 1 y 2

Aunque primero se mande a una pantalla y seguidamente a la otra, al ojo humano la animación se ejecuta al mismo tiempo.

Este protocolo serie también es el utilizado en el caso de querer reproducir vídeos, puesto que también se encuentran en la tarjeta de memoria.

3.7 OTROS PROTOCOLOS SERIE

En el caso de no querer usar librerías para facilitar el manejo de las pantallas, se puede tener la misma funcionalidad siguiendo los protocolos serie propios de las pantallas. Se mandan comandos a través del puerto serie, usando la función propia de Arduino, que se modifica según usemos el puerto serie 1 o el 2 (*Serial1.println()* o *Serial2.printl()*).

Cada protocolo serie cuenta con un comando de inicio específico, que es el que indica a la pantalla qué funcionalidad debe usar, además de unos argumentos.

Los protocolos existentes se pueden dividir en 4:

3.7.1. COMANDOS GENERALES

En este apartado podemos encontrar acciones generales relacionadas con las pantallas, cada una con un comando específico, como son:

- Limpiar la pantalla (45hex): vacía la pantalla. No es necesario ningún comando más.
- Funciones de la pantalla (59hex): aquí se encuentran las funciones tales como encenderse o apagarse o cambiar el rango del contraste. Es necesario introducir también el modo y el valor.
- Sleep (5Ahex): para entrar en modo *sleep*. Es necesario que le acompañen el modo y el tiempo de espera.
- Sonido (4Ehex): para reproducir un sonido. Se incluye pista y duración.
- Configuración de joystick (6Ahex). Es necesario incluir una opción.

3.7.2. COMANDOS DE GRÁFICOS

Aquí podemos encontrar lo necesario para crear gráficos en la pantalla. Cada protocolo consta de un primer comando, que irá seguido de coordenadas, texto, longitudes, dependiendo del protocolo que se use. Entre algunos, están los siguientes protocolos:

- Dibujar un círculo (43hex): se completa con las coordenadas del centro, el radio y el color.

- Dibujar un triángulo (47hex): debe contener los 3 pares de coordenadas correspondientes con los 3 vértices del triángulo, y el color.
- Dibujar una línea (4Chex): con coordenadas de inicio y de final de la línea, y el color.
- Copiar o pegar en la pantalla (63hex): coordenadas de inicio y final del área que se va a copiar, altura y anchura.
- Dibujar un polígono (67hex): número de vértices, sus correspondientes coordenadas, y el color.
- Dibujar un rectángulo (72hex): coordenada superior izquierda e inferior derecha, y color.

3.7.3. COMANDOS DE TEXTO

Como estas pantallas tienen la opción de escritura, a continuación se muestran protocolos para poder llevarla a cabo:

- Elegir tamaño de la fuente (46hex): se completa con el tamaño elegido.
- Hacer texto opaco o transparente (4Fhex): le sigue la opción elegida.
- Escribir una cadena de caracteres (53hex): además del comando propio, continua con las coordenadas donde se inicia la escritura, fuente, color, anchura, altura, cadena de caracteres, y fin de cadena.
- Escribir un carácter ASCII (54hex): se necesita el carácter de la tabla ASCII a escribir, columna, fila y color que va a tener.

3.7.4. COMANDOS PARA LA TARJETA DE MEMORIA

En este apartado aparecen los protocolos serie que obtienen las imágenes, vídeos o animaciones de la tarjeta de memoria micro SD introducida dentro de las pantallas. Aquí podemos encontrar el utilizado en este proyecto. El símbolo '@' (40hex) que aparece en todos los protocolos de este apartado indican una extensión de comando, es decir, que se va a usar el almacenamiento en la tarjeta micro SD:

- Apuntar a una dirección de memoria (@41hex): se debe introducir la dirección de memoria a la que se quiere apuntar.
- Copiar o guardar un pantallazo en la tarjeta de memoria (@43hex): se necesitan las coordenadas del área a copiar, así como su ancho y altura, y la dirección de memoria donde se quiere guardar el área copiada.

- Mostrar imagen o icono desde la tarjeta de memoria (@49hex): coordenadas de inicio, ancho y altura que se la va a dar a la imagen, el color y la dirección de memoria en la que se encuentra guardada.
- Mostrar animación o vídeo (@56hex): es el protocolo utilizado en este proyecto, y se ha explicado anteriormente.
- Inicializar la tarjeta de memoria (@69hex): no necesita más valores añadidos.

Capítulo 4: DEFINICIÓN DEL PROTOCOLO SERIE

4.1. INTRODUCCIÓN

La comunicación serie realiza la transferencia de información enviando o recibiendo datos descompuestos en bits, los cuales viajan secuencialmente uno tras otro.

Uno de los objetivos de este proyecto es el estudio de protocolos serie que sirvan como base para la elaboración de la nueva propuesta de protocolo.

Se ha definido un protocolo serie para comunicar a Arduino lo que debe mandar a las pantallas, siguiendo a su vez otro protocolo, ya establecido: el propio de las pantallas, que se ha definido anteriormente.

4.2. PROTOCOLO SERIE ARDUINO

4.2.1. DEFINICIÓN

Este protocolo surge de la necesidad de ordenar y agrupar los datos que se van a introducir por la aplicación cliente hacia el Arduino. Éste después controlará su entorno a través de los pines digitales, analógicos o puertos serie. Por eso la primera parte del protocolo será definida con vistas a cuál va a ser la función que va a desempeñar el Arduino. Dependiendo de la función se necesitará un protocolo u otro. El definido para este proyecto se explica a continuación.

4.2.2. FUNCIONALIDADES

Esta parte del protocolo serie se hace pensando en futuras ampliaciones del proyecto, incluyendo otras funcionalidades a parte de la conexión de pantallas al Arduino, como pueden ser lectura de los pines digitales, analógicos o generación de una señal PWM. Debido a esto, el inicio del protocolo serie define a qué función está referida, quedando las siguientes posibilidades:

- DP: se trata de la función para los displays o pantallas.
- DI: entradas y salidas digitales.
- AN: entradas y salidas analógicas.
- PW: generar señales PWM.

Una vez recibido el comando por puerto serie, al procesarlo se comprueba que en este caso sea para la función DP. Es nuevamente comprobado en el programa principal.

Lo que sigue en el protocolo serie solo es funcional en el caso de la función para los displays, puesto que para las otras habría que introducir otros valores.

4.2.3. DESCRIPCIÓN DE LA TRAMA

Una vez elegida la función DP, el protocolo serie consta de otros 6 bytes (de un total de 8). Estos bytes deben introducirse en orden con unos valores determinados para indicar a las pantallas qué deben ejecutar y qué se debe hacer después. El tiempo máximo que se permite transcurrir entre comando y comando es de 2 segundos. Este tiempo se puede variar en la librería **available.h**. Si han transcurrido los 2 segundos, se da a entender que se ha terminado de introducir la trama y se pasaría al procesado de los bytes. Estos bytes son:

Byte 3: ojo o pantalla

Indica qué pantalla o pantallas van a ser las encargadas de ejecutar la animación. Se ha estipulado lo siguiente:

- 1: pantalla izquierda. Corresponde con el puerto serie 1 del Arduino.
- 2: pantalla derecha. Corresponde con el puerto serie 2 del Arduino.
- 0: las dos pantallas. Se usan los puertos serie 1 y 2.

En el programa queda almacenado en la variable **ojo.**

Esta especificación de la pantalla se realiza por el motivo de que cada pantalla va unida a un puerto serie distinto. Debido a esto, a la hora de mandar los datos a las pantallas se chequea la variable ojo para comprobar a través de qué puerto serie se hace. En el caso de ser a las dos pantallas (opción 0), primero se envía por un puerto y luego por el otro.

Byte 4 y 5: animación

Este byte surgió de la necesidad de especificar qué expresión debe mostrar el robot. Determina el número de animación que se va a ejecutar (sonreír, enfadarse, parpadeo). Hay un total de 15 animaciones, todas ellas emulan a expresiones humanas. Algunas de ellas van emparejadas porque son los opuestos, referido a deshacer la animación, como por ejemplo sonreír y dejar de sonreír. El número concreto de cada animación está recogido en la

Tabla 4.

Número	Animación
00	Parpadear
01	Sonreír
02	Dejar de sonreír
03	Sorprenderse
04	Dejar de sorprenderse
05	Entristecerse
06	Dejar de entristecerse
07	Sospechar
08	Normal
09	Enfadándose
10	Desenfadándose
11	Abriendo medio b
12	Cerrando medio b
13	Abriendo medio
14	Cerrando medio

Tabla 4: Animaciones disponibles

Cada animación tiene una dirección de memoria propia y tarda un tiempo distinto en ejecutarse. Estas características se encuentran en la librería *valores.h* y pueden ser cambiadas sin necesidad de alterar al código principal, en el caso de querer cambiar las animaciones o su orden.

En el código principal, quedaría guardado en la variable **gif**, que será la utilizada para hacer una selección del tipo *switch case*, puesto que cada animación va a tener sus propios parámetros.

Algunas de las animaciones se muestran en la Figura 40.

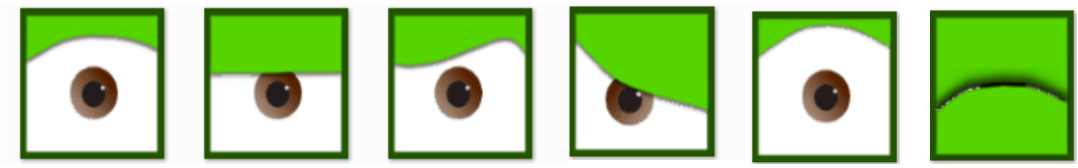

Figura 40: Expresiones

Byte 6: tiempo

Este byte es necesario ya que las transiciones entre animaciones pueden no quedar limpias. Debido a la existencia de este byte, existe la posibilidad de añadir un tiempo extra de espera una vez ejecutada la animación. Durante el transcurso de este tiempo se mantiene en pantalla la última imagen de la animación antes de ejecutar la siguiente. Esto se hace con el propósito de que el tránsito entre una y otra sea más natural. Por ejemplo, en el parpadeo, sin esta opción parpadearía cada 1520ms (cada segundo y medio aproximadamente). Sin embargo si le ponemos un tiempo extra al final de cada parpadeo, quedaría más real, más parecido a un humano.

Se introduce en segundos, y en el caso de que no se quiera añadir tiempo extra este byte deberá tener el valor 0. De no ser así, saltaría el error de comando, puesto que detecta que falta uno de los bytes por introducir.

Si durante este tiempo extra llega un comando nuevo por puerto serie se pasaría a ejecutarlo, es decir, se reduce este tiempo extra o incluso se elimina. En el caso del parpadeo se fuerza a que este tiempo siempre sea de dos segundos.

En el código se almacena en la variable **tiempoEXTRA**.

Byte 7: estado

Una vez ejecutada la animación, falta por concretar qué se debe hacer después. Así surge este séptimo byte, en el que se determina lo que se tiene que hacer cuando se ejecute la animación. Existen 5 posibilidades que son las siguientes:

- **1**: hacer el opuesto. Como se ha mostrado, varias de las animaciones están emparejadas con su opuesto. Así pues, si se ha mandado ejecutar por ejemplo la animación de sonreír, si se encuentra en este estado, pasaría a ejecutar el opuesto (dejar de sonreír).
  En el caso en el que la animación no tenga opuesto y se encuentre en este estado, pasa a la animación de "normal".
  Después de ejecutarse el opuesto o la animación de "normal", vuelve a parpadear.

- **2**: repetición no infinita. En este estado se repite la animación ejecutada anteriormente hasta que llega nuevo comando por puerto serie. En este caso, se ejecuta la nueva acción.

- **3**: repetir siempre. La diferencia con el estado anterior es que al llegar un nuevo comando y ejecutarlo, se vuelve a la animación anterior, a no ser que el nuevo estado también sea 3, en cuyo caso esta nueva animación sería la que se estaría repitiendo siempre.

- **4**: espera. Se mantiene en la pantalla la última imagen de la animación ejecutada hasta nuevo comando. Sea cual sea este nuevo comando, primero se ejecuta el opuesto de la animación anterior. Así, si teníamos por ejemplo la animación de sonreír, primero deja de sonreír y luego se ejecuta el nuevo comando.

- **5**: únicamente se usa cuando se acaba de pasar por el estado 4 y cuando lo que se manda en el opuesto. Es decir, para quitar el estado 4 mandas el opuesto, éste se ejecuta y vuelve a parpadear.

En el código queda representado por la variable **repetir**.

Byte 8: retorno de carro

Al utilizar la aplicación cliente y no la monitorización serie propia del Arduino, este último byte se añade de forma que así se tenga notificación de que se ha enviado una trama completa. En el caso de que este último byte no se corresponda con el retorno de carro, dará error, puesto que indica que o se ha mandado una trama corta o larga, o que no se ha dado la información de que se ha terminado de mandar la trama.

Además, si el algún momento queremos cambiar la aplicación cliente y usar el hyperterminal de Windows, al pulsar el intro ya pasa a formar parte de la trama, por lo que el procesado sería el mismo para ambas formas de introducir los datos y no habría que hacer ningún cambio.

4.2.4. EJEMPLO

A continuación se muestran 2 ejemplos para una mejor comprensión del protocolo serie:

- Ejemplo 1: en este ejemplo lo que se pretende que se ejecute en las pantallas es que permanezca parpadeando por ambos displays continuamente. Para ello se introduce la trama mostrada en la Tabla 5.

BYTE	1	2	3	4	5	6	7	8
VALOR	D	P	0	0	0	2	3	INTRO
SIGNIFICADO	Se trata de la función para los displays		Las dos pantallas serán las encargadas de ejecutar la animación	Animación número 0, correspondiente al parpadeo		Se añaden dos segundos como tiempo extra	Se va a repetir siempre	Retorno de carro, fin de la trama

Tabla 5: Protocolo serie, ejemplo de trama 1

- Ejemplo 2: en este segundo ejemplo se desea que, únicamente por la pantalla izquierda, se ejecute la animación de sonreír, para que, una vez sonreído, vuelva a su estado normal dejando de sonreír. Para lograr esto cada byte del protocolo serie debe tener los valores mostrados en la Tabla 6.

BYTE	1	2	3	4	5	6	7	8
VALOR	D	P	1	0	1	0	1	INTRO
SIGNIFICADO	Se trata de la función para los displays		La pantalla izquierda será la que ejecute la animación	Animación número 1, correspondiente a la de sonreír		No se añade tiempo extra	Se va a ejecutar después el opuesto	Retorno de carro, fin de la trama

Tabla 6: Protocolo serie, ejemplo de trama 2

4.2.5. IMPLEMENTACIÓN SOFTWARE. REQUISITOS

La implementación de este protocolo serie se va a realizar en lenguaje C, pero dentro del entorno de programación de Arduino, por lo que se deben de tener en cuenta sus funciones propias, como las de lectura del puerto serie o imprimir por puerto serie.

El programa debe de incluir una función de lectura del puerto serie para poder obtener la trama del protocolo serie una vez que ha sido enviada a Arduino.

También debe constar de una función que imite a un temporizador, para que así Arduino sepa que, pasado el tiempo estipulado, se ha terminado te mandar la trama, y por lo tanto debe ser procesada.

De esto último se obtiene la necesidad de una función de procesado de la trama, para comprobar que los datos que se han recibido están dentro de los límites esperados, y de que se han enviado todos, puesto que se comprueba el inicio y el final de trama.

Por último, en el programa se debe incluir una función a la cual se pueda acudir en el caso de que el protocolo serie haya sido introducido erróneamente, o de manera incompleta.

Todas estas funciones aseguran la correcta recepción y procesado del protocolo serie. Se deben encapsular junto con el resto de funciones de lectura de los otros puertos serie en una librería.

Capítulo 5: IMPLEMENTACIÓN

5.1. INTRODUCCIÓN

Para llevar a cabo la implementación de este proyecto se utilizan recursos de software y hardware, que combinados, consiguen desarrollar el sistema requerido en este proyecto.

5.2. IMPLEMENTACIÓN HARDWARE

5.2.1. INTRODUCCIÓN

A parte de los códigos y librerías, es necesaria una implementación hardware, puesto que al tratarse de varios dispositivos debe existir una conexión física entre ellos. Así tendremos una conexión del ordenador al Arduino, y otra del Arduino a las pantallas.

5.2.2. CONEXIÓN DE PLACA ARDUINO A LAS PANTALLAS

A parte de la placa Arduino y de las dos pantallas, es necesaria una unión entre ambas. Esta unión ya está predefinida. Existe la posibilidad de acoplar directamente la pantalla con la placa, de manera que Arduino no pierda sus pines, como se puede observar en la Figura 41.

Figura 41: Arduino + placa shield

En este caso, a parte de que las pantallas no estarán accesibles (irán incorporadas a un robot), se trata de usar dos pantallas usando un único Arduino, y las 2 pantallas no pueden ir acopladas usando una placa shield. Por este motivo se tienen que conectar punto a punto a través de cables como se muestra en la Figura 42.

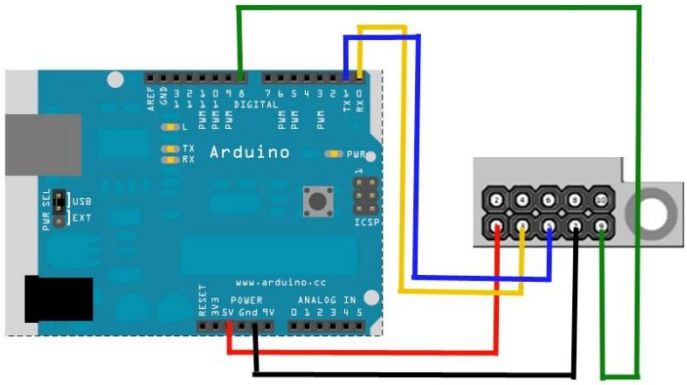

Figura 42: Cableado pantalla + Arduino

El trazo verde es el pin de reseteo de la pantalla. Debe ir conectado al pin digital que se ha establecido para tal función. En este caso no se ha elegido ninguno.

Los trazos amarillo y azul unen los pines correspondientes al puerto serie. Se conectarán tanto los TX como los RX, puesto que los displays elegidos a parte de recibir por puerto serie, también envían. En este proyecto las pantallas irán unidas al puerto serie 1 y 2 (TX1-RX1 y TX2-RX2). Al usarse el Arduino Mega 2560, quedaría un tercer puerto serie sin utilizar.

Los trazos negro y rojo corresponden a la tierra y alimentación respectivamente. Las pantallas se alimentan con un voltaje de 5 voltios, por lo que se puede sacar directamente la conexión de la placa Arduino. Dicha placa obtiene ese voltaje de su conexión con el ordenador.

Estas conexiones se tendrán por duplicado, puesto que se va a trabajar con dos pantallas. No será necesaria una placa protoboard auxiliar puesto que la placa elegida de Arduino tiene varias salidas de 5V y de GND.

En la Figura 43 se pueden observar los cables que han sido soldados para poder realizar la conexión con las pantallas y Arduino. En un extremo los pines están unidos, puesto que serán los que irán unidos a la pantalla; en el otro extremo están sueltos, ya que en la placa Arduino cada pin se conecta en un sitio, no están juntos.

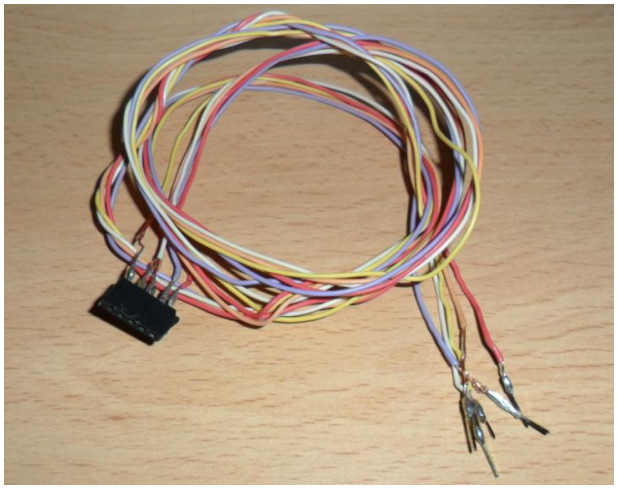

Figura 43: Cables de conexión

En la Figura 44, se encuentra una de las pantallas uOLED con la conexión realizada. La imagen que muestra la pantalla es la de inicio propia.

Figura 44: Conexión vista desde delante

En la Figura 45, las dos pantallas se encuentran montadas en un armazón y ambas están cableadas listas para su conexión con la placa Arduino.

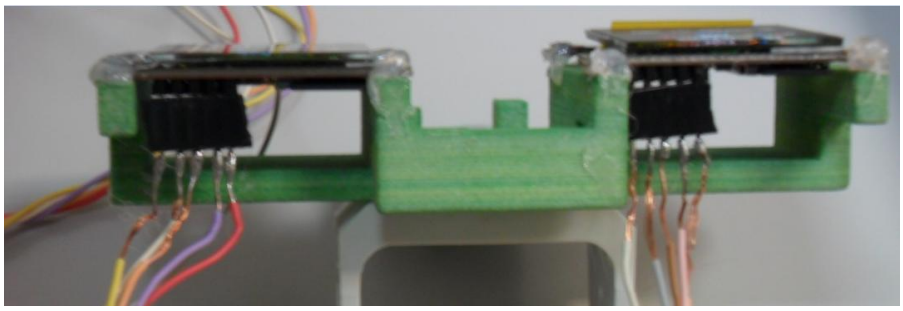

Figura 45: Conexión vista desde atrás

Una vez que están las pantallas listas para ser conectadas se procede a cablearlas hacia Arduino. La conexión más agrupada queda en los puertos serie. En la Figura 46 se puede observar que están ambas pantallas conectadas. La izquierda al puerto serie 1, usando los pines TX1 y RX1, y la derecha al puerto serie 2, usando los pines TX2 y RX2.

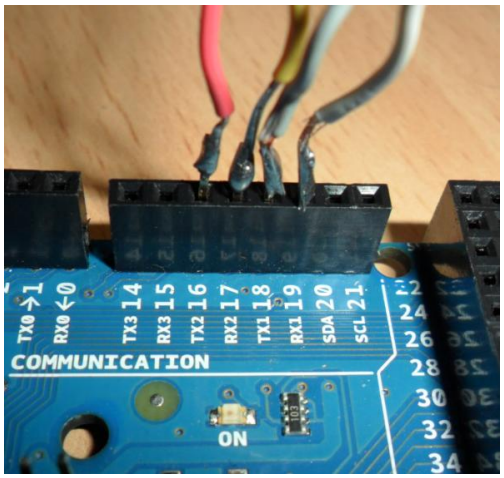

Figura 46: Conexión puertos serie

Una vez conectados los puertos serie, se pasa a la conexión con la alimentación y tierra. En la Figura 47 únicamente aparece una de las pantallas conectadas, puesto que la otra se conecta en otros pines de +5V y GND, que se encuentran junto a los pines digitales. De no existir estos pines, se debería usar una placa protoboard para multiplicar los pines disponibles.

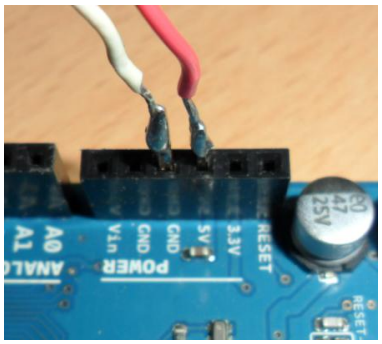

Figura 47: Conexión a la alimentación y tierra

El voltaje que usa la placa Arduino para alimentar las pantallas, como se ha mencionado anteriormente, proviene de su conexión con el ordenador.

5.2.3. CONEXIÓN DE LA PLACA ARDUINO CON EL ORDENADOR

Esta conexión de Arduino con el ordenador se realiza a través de un cable USB 2.0 AM/BM. No es un cable USB básico (AM/AM), si no que uno de los extremos tiene una conexión BM, que será el que irá conectado a la placa Arduino. La diferencia entre un extremo y otro se muestra en la Figura 48.

Figura 48: Cable USB 2.0 AM/BM

A través de esta conexión la placa Arduino queda totalmente alimentada, alimentando a su vez a las pantallas usando sus pines para tal función. Este cable también representa la conexión de Arduino y ordenador a través del puerto serie 0, ya que realiza tanto la función de alimentación como la de comunicación serie.

En la Figura 49 se muestra la placa ya conectada al ordenador, quedando totalmente alimentada.

Figura 49: Conexión de la placa Arduino con el ordenador

5.2.4. CONEXIÓN ORDENADOR-ARDUINO-PANTALLAS

Como se ha mencionado anteriormente, una vez realizada la conexión de la placa Arduino con el ordenador, ésta ya tiene disponible sus pines de alimentación, que son los utilizados para alimentar a las pantallas. Por lo que una vez realizadas todas las conexiones, Arduino y pantallas estarían encendidas y operativas, tal como se observa en la Figura 50.

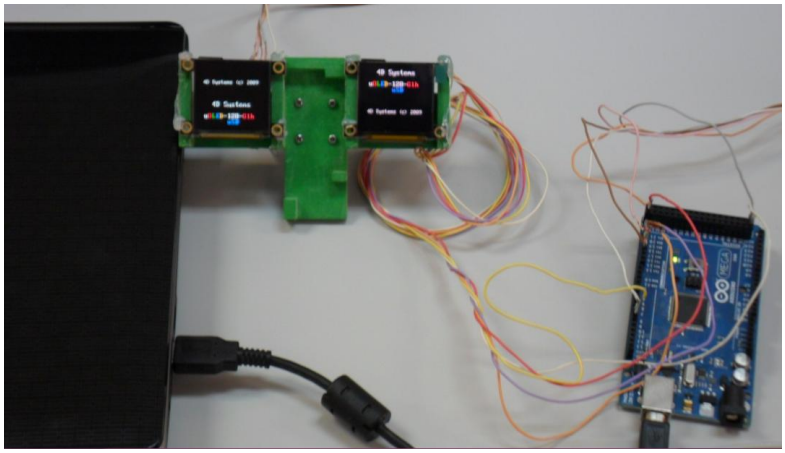

Figura 50: Conexión ordenador+Arduino+pantallas

5.3. FLUJOGRAMAS

Para la explicación del programa implementado se han creado unos flujogramas con su descripción que muestran cómo está estructurado el programa.

En la Figura 51 se muestra el flujograma correspondiente al programa principal.

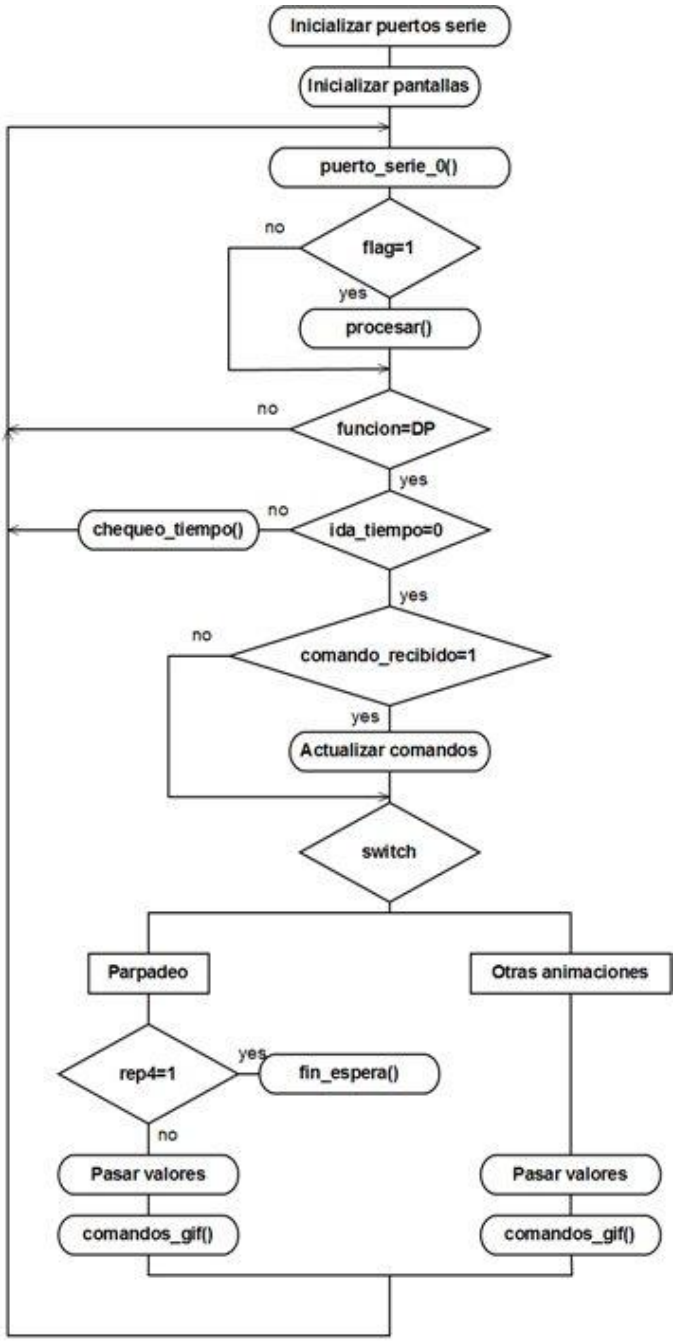

Figura 51: Flujograma principal

Lo primero que hace el programa es inicializar los puertos serie (velocidad en baudios) e inicializar las pantallas (ver Figura 52). Esto sólo se hace una vez, cuando se manda ejecutar el programa, ya que estos comandos se encuentran fuera del bucle.

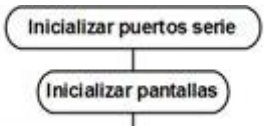

Figura 52: Inicialización

A continuación empieza el bucle principal. Se pasa a leer el puerto serie 0 a través de la función *puerto_serie_0()*. Esta función devuelve la variable **flag**, que se activa cuando ha llegado un comando nuevo. Por eso, cuando está activada se pasa a procesar el comando a la función *procesar()* (ver Figura 53).

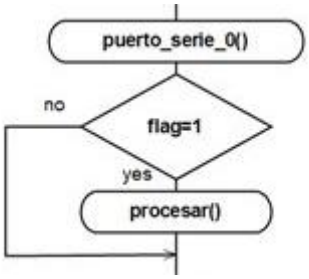

Figura 53: Lectura puerto serie

Una vez llegado a este punto, se comprueba si el programa debe meterse a realizar las operaciones propias de la función referida a los displays. En caso afirmativo, se pregunta si se está ejecutando alguna animación para no solaparlas a través de la variable **ida_tiempo**. Es decir, siempre se espera el tiempo mínimo necesario para que se ejecute la animación ordenada anteriormente.

Si esta variable no tiene constancia de que haya pasado el tiempo, el programa se dirige a la función *chequeo_tiempo()* para saber si ya ha terminado de ejecutarse. Si por otro lado, la variable **ida_tiempo** nos dice que ya ha terminado la ejecución, se pasa a preguntar si ha llegado un comando nuevo. Para ello se utiliza la variable **comando_recibido**, que se activa cuando **flag** es positiva. Si hay comando nuevo se pasan los nuevos parámetros, si no se conservan los anteriores. Este proceso se observa en la Figura 54.

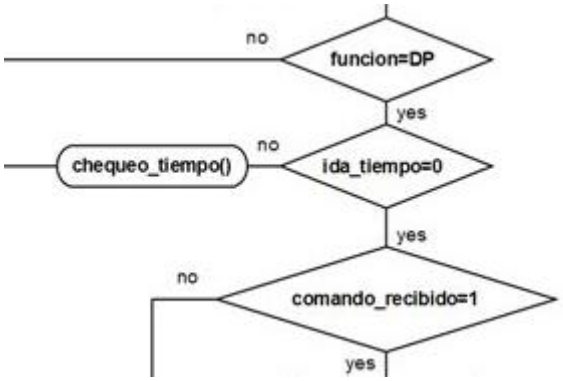

Figura 54: Función DP

Una vez actualizadas las variables con el nuevo comando, o manteniendo las anteriores si no ha llegado ningún comando nuevo, se pasa al switch (ver Figura 55) en el que la variable es la animación (**gif**).

El único caso en el que se hace algo distinto es en la animación de **'parpadeo'**. Lo primero que hace el programa es forzar a que **repetir** sea 3 (que se repita siempre) y que **ojo** sea 0 (las dos pantallas).

Debido al byte 7 del protocolo serie, como se ha explicado anteriormente, existe la opción en la que se mantiene fija la última imagen de la animación hasta nuevo comando (opción 4), que podía ser el opuesto o parpadear. Si llega a **'parpadeo**' a través de esta opción, pasará por la función *fin_espera()*. Si no se había seleccionado esta opción, se actuará como con el resto de los casos o animaciones.

La parte común a todas las animaciones tiene varias instrucciones. Primero, se pasan los valores correspondientes a la animación seleccionada en el switch. Estos valores son los parámetros que hay que introducir en la trama de protocolo serie que irá dirigida a las pantallas y el opuesto de la animación en caso de tenerlo.

Una vez pasados estos datos, se irá a la función *comandos_gif()* que es la encargada de mandar por puerto serie la trama necesaria a las pantallas para ejecutar las animaciones.

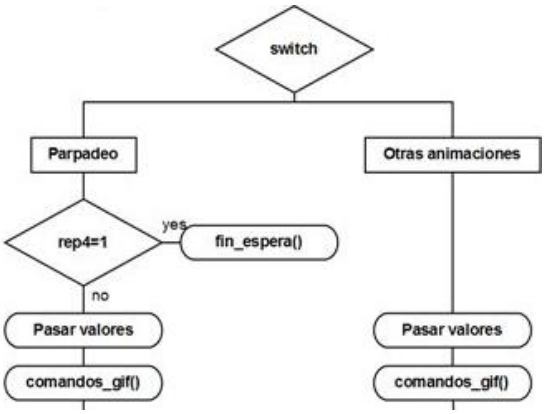

Figura 55: Switch case

Tras esto se vuelve a leer el puerto serie 0, es decir, se vuelve a empezar el bucle.

En los apartados que siguen se explican las funciones implementadas.

5.3.1. PUERTO_SERIE_0

Esta función (ver Figura 56) se encarga de leer el puerto serie 0, el que recibe los comandos. Primero comprueba que hay datos disponibles para leer en el puerto serie. Si es así, lo lee y almacena en la variable **comando[]**, y avanza por ella a través de un contador. Se incrementa el contador para el siguiente dato y se inicializa el temporizador.

Si no hay datos disponibles, pregunta si se han recibido anteriormente con la variable **com**, para ir a la función *tiempo()* a comprobar si ha pasado el tiempo necesario para saber si se ha terminado de mandar el comando.

Si **com** tiene valor cero, fuerza a desactivar la bandera para indicar que no hay comando nuevo recibido.

Independientemente del camino seguido, al final se devuelve el valor de **flag** al programa principal.

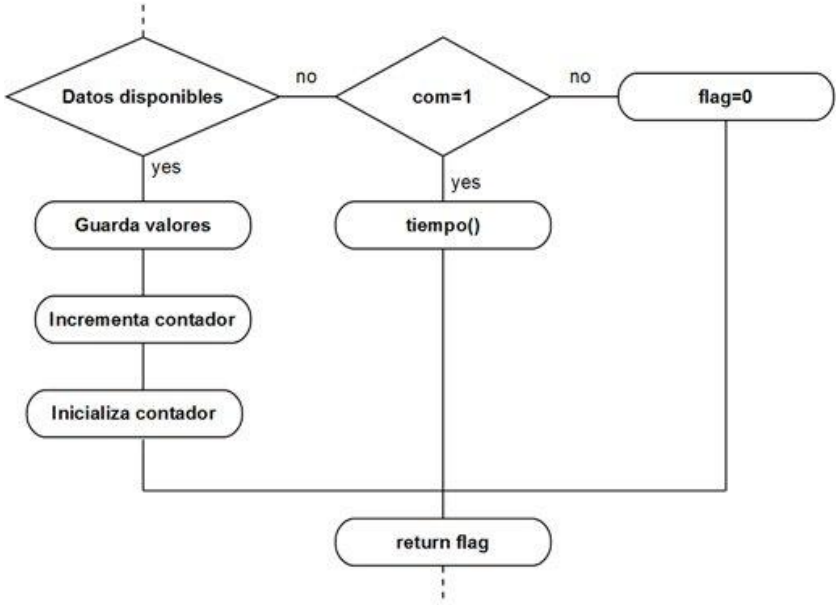

Figura 56: Función puerto_serie_0()

5.3.2. TIEMPO

Esta función (ver Figura 57) actúa como temporizador. Se encuentra dentro de la librería *available.h*.

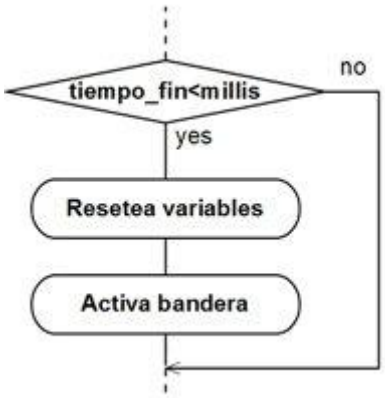

Figura 57: Función tiempo()

Cuando se confirma que ha pasado el tiempo, en este caso 2 segundos, quiere decir que se ha terminado de recibir un comando nuevo. Se resetea el contador usado para guardar los comandos (variable **i**) y la variable **flag** se activa, para luego saber que se ha recibido un comando nuevo.

El contador usado es la función *millis()*, que va almacenando los milisegundos que han pasado desde el inicio del programa. Es propia del entorno Arduino y se explicará más adelante.

5.3.3. PROCESAR

Una vez que se ha recibido el comando entero, hay que comprobar su longitud y si se refiere a la función de los displays, puesto que es la única función que se ha implementado en este proyecto.

Si el caso es afirmativo, se comprueba que el resto de valores se encuentran dentro de sus intervalos. Si es así, se pasa a descomponer el comando, según el protocolo serie establecido. Esto es, el byte 3 será el valor de la variable que contenga la pantalla o pantallas que van a ejecutar la animación, los bytes 4 y 5 serán la animación, y así consecutivamente con todo el comando. Una vez pasadas todas las variables, se resetea la variable **comando[]**.

Si el comando recibido no tiene la longitud adecuada o no se refiere a la función de los displays, o los valores no están dentro de sus intervalos, se va a la función *error()*. Todo esto queda representado en la Figura 58.

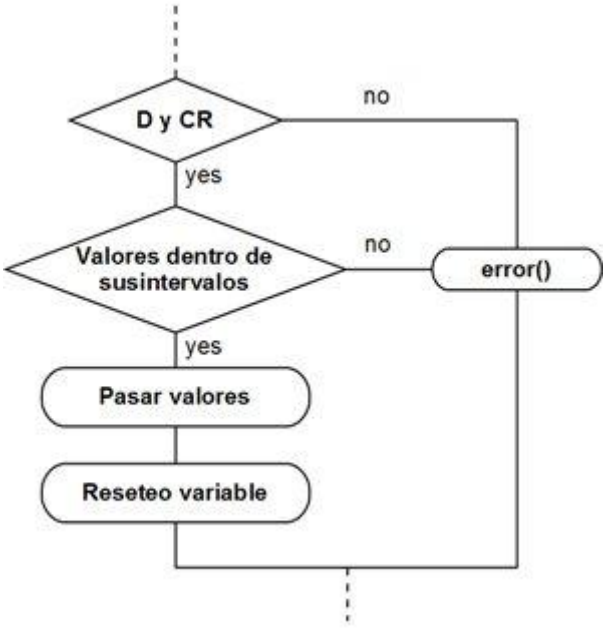

Figura 58: Función procesar

5.3.4. ERROR

Lo que hace esta función (ver Figura 59) es imprimir por pantalla el mensaje de "error" para hacer saber que ha ocurrido algún error al introducir el comando o que los valores no son los correctos. Tras esto, resetea los valores de **comando[]** para que al introducir los nuevos datos no se solapen.

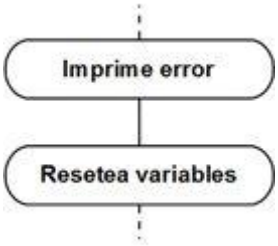

Figura 59: Función error

5.3.5. FIN_ESPERA

Según la Figura 60, una vez que se encuentra en esta función se pasan los valores necesarios para que el programa sepa qué es lo que tiene que hacer a continuación. Así, se modifican las variables **gif** y **siempre**. La primera es para saber qué animación es la que se debe ejecutar inmediatamente después (en este caso, el opuesto), y la segunda es para saber qué animación se ejecuta tras el opuesto, que será el **'parpadeo'.**

Figura 60: Función fin_ espera

5.3.6. COMANDOS_GIF

Lo primero que hace esta función (ver Figura 61) una vez actualizadas las variables con los valores propios de la animación, es procesarlos.

La dirección de memoria se divide en tres segmentos (baja, media y alta) y en número de imágenes en 2 (baja y media). En realidad el número de imágenes también se divide en 3, pero la parte alta es común a todas. Esta división de hace usando una máscara.

Cuando los datos estén procesados, se comprueba a qué ojo u ojos van a ser mandados, puesto que varía el puerto serie. En el caso de que se manden a los dos ojos, primero se mandan a uno y luego al otro. Este proceso se hace tan rápido que no es perceptible al ojo humano la diferencia de tiempo entre uno y otro.

Para mandar los comandos se usa la función *Serial1.print()* o *Serial2.print()*, dependiendo del puerto serie utilizado.

En el momento en el que los datos han sido mandados, se inicializa el temporizador usando la función *millis()*, que recoge el tiempo transcurrido en milisegundos desde que se inicia el programa.

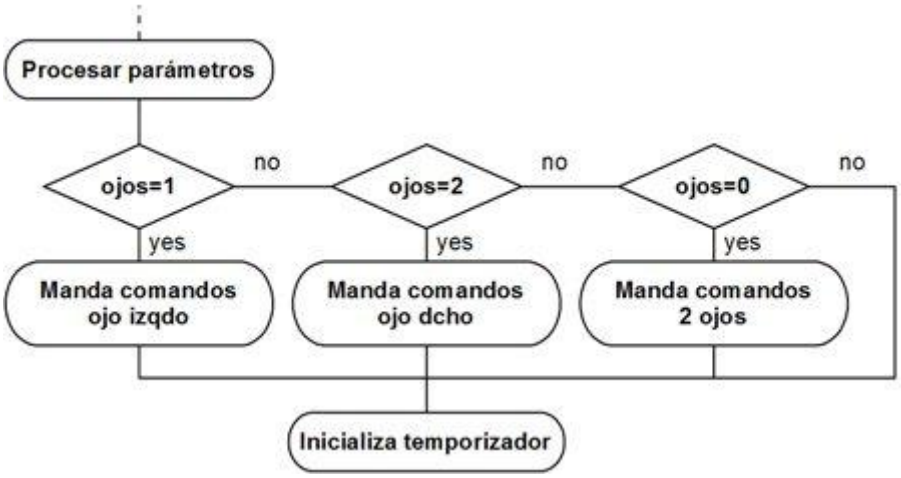

Figura 61: Función comandos_gif

5.3.7. CHEQUEO_TIEMPO

Esta función (ver Figura 62) lo primero que pregunta es si ha pasado el tiempo necesario para que se ejecute la animación (tiempo_FIN<millis()) y si además se ha recibido un comando nuevo (recibido=1).

El motivo de esta pregunta es que si se ha recibido un nuevo comando, no hace falta esperar el tiempo extra introducido en el protocolo serie.

Si la respuesta es afirmativa, pasa a leer los puertos serie 1 y 2 para comprobar si las pantallas han ejecutado la animación. Si se ha ejecutado, se va a la función *condiciones()* para ver qué se hace después. Si no se ha ejecutado, las pantallas mandan un **nack**, de valor 15, por lo que el programa se dirigirá a la función *error_ack()*. Cuando se recurre a esta función lo normal es que haya ocurrido un error en las pantallas.

Si por otro lado, no se ha recibido ningún comando nuevo, se pregunta si ha pasado el tiempo necesario más el tiempo extra añadido. Si es así, el procedimiento a seguir es el mismo que el caso anterior.

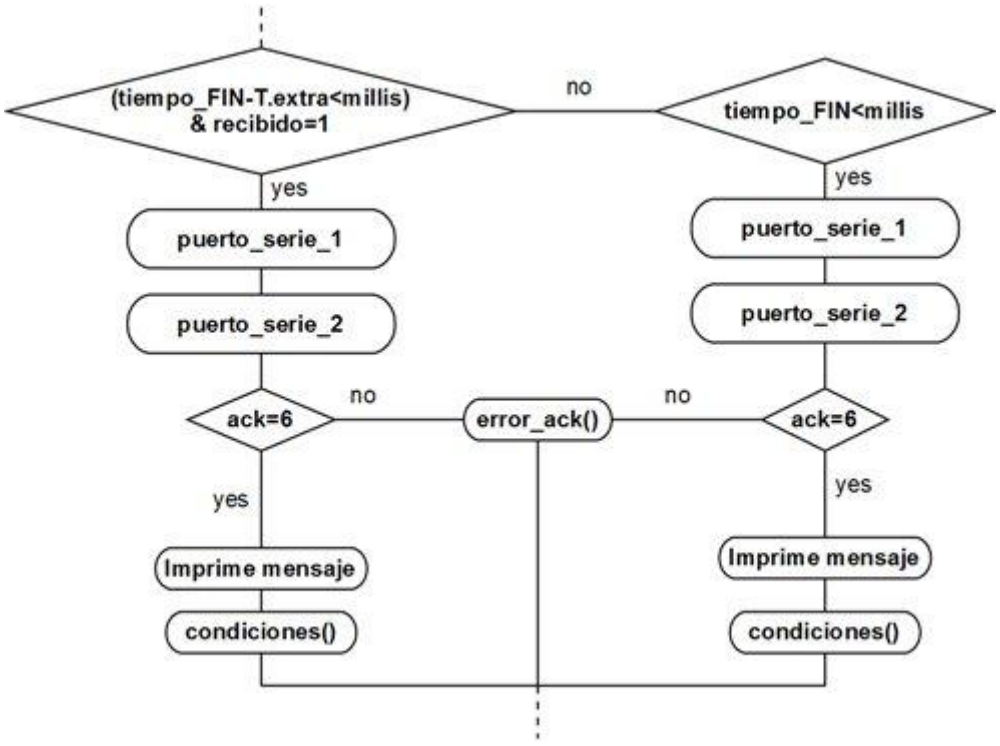

Figura 62: Función chequeo_tiempo

5.3.8. ERROR_ACK

Se llega a esta función en el caso de que no se haya ejecutado la animación por algún problema de la pantalla, por lo que, como se muestra en la Figura 63, imprime por pantalla las respuestas recibidas por ambas pantallas, puesto que si sólo es una de ellas la que ha dado el error, no haría falta reiniciar a ambas. Esto se puede ver en el monitor serie de Arduino. El programa continua ejecutándose con normalidad.

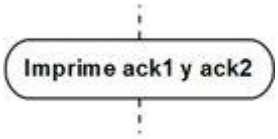

Figura 63: Función error_ack

5.3.9. PUERTO_SERIE_1 Y PUERTO_SERIE_2

Al igual que con la lectura del puerto serie 0, lo primero es preguntar si hay datos disponibles para leer, como se puede observar en la Figura 64. De ser así, se almacena el dato en la variable **ack1** o **ack2**, dependiendo del puerto serie leído. Estas variables serán las que nos indiquen si la animación se ha ejecutado correctamente o ha ocurrido algún problema.

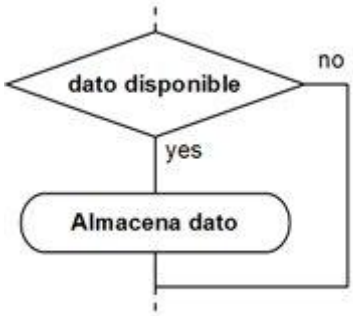

Figura 64: Funciones puerto_serie_1 y puerto_serie_2

5.3.10. CONDICIONES

Como muestra la Figura 65, en esta función se atiende al byte 7 del protocolo serie, que aquí queda representado por la variable **rep**. La variable **gif** es la animación que se tiene que ejecutar inmediatamente después.

Esta función surge de la necesidad de saber qué se debe hacer una vez ejecutada la animación. Existen varias posibilidades:

- La primera cuestión es si **rep=1** y además se ha ejecutado ya el opuesto, con la variable **opo**. Si es así, pasa a parpadear.

- Si no es así, comprueba si está en la opción 1, y entonces pasa a ejecutar el opuesto, guardado en la variable **caso_2**, y activando la variable **opo**.

- Si por el contrario se encuentra en la opción 2, **gif** no va a variar su valor anterior.

- Si es la 3, la variable **siempre** se actualiza con la animación recién ejecutada, para que se esté ejecutando continuamente.

- Si por último, es la opción 4, se activa la variable **rep4** para indicar que se está en esta opción. Se cambia el valor de **miD**, que es la variable que verifica que la función es para los displays, para que no se acceda a esta función y el programa lo único que haga sea leer el puerto serie 0.

- En el caso de ser la 5, quiere decir que está saliendo del estado 4. Una vez en esta opción, a **gif** se le da el valor guardado en **siempre**. En esta variable suele encontrarse la animación de **'parpadeo'**, ya que en esta animación se fuerza a ello.

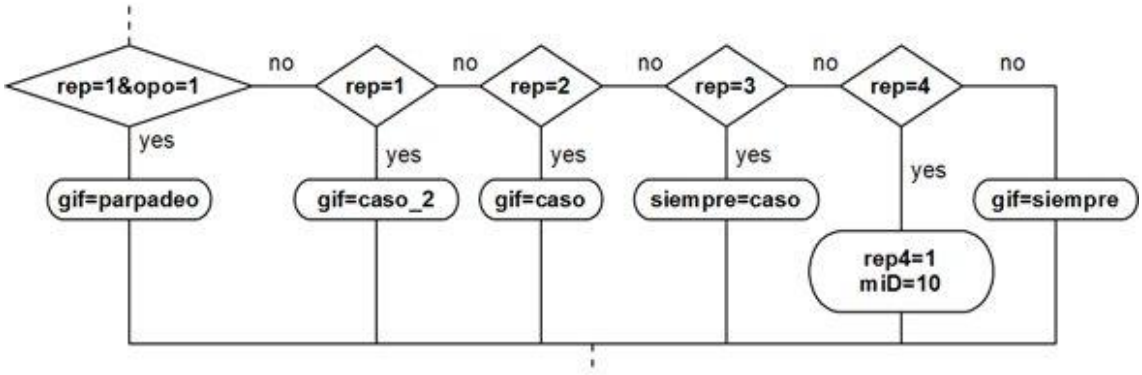

Figura 65: Función condiciones

5.4. IMPLEMENTACIÓN SOFTWARE

5.4.1. INTRODUCCIÓN

La implementación escrita consta del programa principal completado con librerías creadas o modificadas especialmente para el proyecto, además del código implementado para la introducción de los comandos (aplicación cliente).

El programa principal está diseñado para que esté constantemente leyendo el puerto serie 0, por donde llega la trama. Esto se hace con el fin de priorizar al comando nuevo recibido, siempre esperando a que termine de ejecutarse el anterior. De este modo no queda ningún comando sin ser leído. En lo respectivo a esperar a que se ejecute la animación anterior, se hace por el motivo de que si esto no se hiciera, se solaparía el protocolo serie mandado a las pantallas y no se ejecutaría ninguno de los 2.

Esta lectura del puerto serie se hace a través de funciones implementadas en una librería. Además, en esta librería también se incluye la lectura de los otros dos puertos serie (el 1 y el 2), los correspondientes a las pantallas.

A pesar de que el entorno Arduino tiene su propio monitor serie, en este proyecto se utiliza la aplicación cliente para mandar (escribir) por puerto serie y recibir (imprimir por pantalla) en el caso de que el comando sea erróneo.

Todo lo recibido de Arduino se puede leer también en el monitor serie, aunque a través de él no se pueden realizar envíos. De la manera en la que está implementado el código, siempre indicaría error de comando, ya que el fin de trama del protocolo serie está marcado por el carácter 'CR', que en el monitor serie no se tiene en cuenta.

5.4.3. LIBRERÍAS

Parte del código ha sido encapsulado en un total de 3 librerías para simplificar el código principal y tener más organizadas las funciones:

- Librería *_oled160drv.h*

Es la librería utilizada para manejar las pantallas. Evita tener que hacer la comunicación de inicializar las pantallas usando el protocolo serie de cada función que se quiera usar. Simplemente hay que introducir los argumentos correspondientes, que en este caso será la

pantalla a la que vaya referida la función. En la web se pueden encontrar distintas librerías para esta finalidad ya implementadas, pero ninguna tiene en cuenta el uso de dos pantallas, por lo que se tienen que realizar modificaciones para usar los dos puertos serie. Estas modificaciones han de hacerse en todas las funciones en las que intervenga el puerto serie.

La original fue creada por Óscar González, y se puede descargar gratuitamente, ya que está subida en la web [17]. En su página web [18] muestra un ejemplo de lo que se puede hacer con la librería, las pantallas y Arduino. En el anexo A.3 se puede observar la librería original y la modificada y empleada finalmente en el proyecto.

- Librería *valores.h*

En esta librería se incluyen todos los *defines* necesarios para dar valor a las constantes empleadas, así como los argumentos necesarios para cada animación: comandos que se mandan a la pantalla y mensaje que se debe imprimir una vez ejecutada la animación.

La finalidad de esta librería es que cualquier cambio que se tenga que realizar, se haga únicamente aquí y no en el programa principal. Es decir, que si hay que cambiar algún valor de una constante se cambia en la librería, facilitando la adaptación al cambio. Esto es útil sobre todo a la hora de cambiar las animaciones, puesto que solo habría que modificar los campos que sufran cambios, como puede ser la dirección de memoria, número de imágenes y tiempo entre ellas, tiempo total que tarda en ejecutarse.

También se puede cambiar el número de la animación para cambiar el orden en caso de que se eliminan o añadan animaciones.

- Librería *available.h*

En esta librería se reúnen las funciones de lectura de los puertos serie y de su posterior procesamiento. Todas están reunidas en una misma clase.

Dentro de la librería se encuentra el archivo *available.cpp* donde se implementan las funciones declaradas en la clase. Estas funciones se explicarán más adelante.

No todas las funciones de esta librería son llamadas desde el programa principal. Únicamente se llama a las que leen los puertos serie 0 (conexión con el ordenador), 1 y 2 (conexión con las pantallas).

El puerto serie 0 se lee para recibir la trama con la información necesaria para saber qué animación se ha de ejecutar y qué hacer después, siguiendo el protocolo serie implementado

para tal causa; los puertos serie 1 y 2 se leen para obtener la respuesta enviada por las pantallas tras la ejecución de cada animación.

5.4.4. FUNCIONES EN EL PROGRAMA PRINCIPAL

Estas funciones se encuentran en el archivo de extensión **.pde**. Están declaradas al inicio del programa e implementadas al final. Los flujogramas correspondientes se pueden observar en el punto anterior. Estas funciones son las siguientes:

- *void condiciones(int rep,int caso,int caso2,int opo)*

En esta función se analiza lo que se debe hacer una vez ejecutada la animación usando la información del protocolo serie y de la animación ejecutada anteriormente.

Los argumentos de esta función son:

- rep: es el status proveniente del comando recibido por puerto serie. Equivalente a **repetir**.
- caso: se trata de la animación que se ha ejecutado.
- caso2: se trata de la animación que se tiene que ejecutar a continuación.
- opo: si está activada indica que se ha realizado ya el opuesto.

- *void comandos_gif(char del,char nframes,long address, int ojos)*

En la función *comandos_gif* se manda a través de los puertos serie 1 y 2 la información necesaria para la ejecución de una animación concreta, como su dirección de memoria, frames y delay. Primero se analiza a qué pantalla o pantallas se deben de mandar, dependiendo del protocolo serie recibido. Una vez mandados los comandos siguiendo el protocolo serie de las pantallas se pone en marcha el temporizador.

Los argumentos de esta función son:

- del: se refiere al delay o tiempo entre frames.
- nframes: frames. Es el número de imágenes que contiene la animación.
- address: es la dirección de memoria en la que se encuentra la animación.

- *void fin_espera (caso2)*

Esta función únicamente se utiliza si se recibe por protocolo serie el status 4, que corresponde a que se mantenga la animación en pantalla una vez ejecutado hasta recibir otro comando. En este caso, este otro comando es el parpadeo y se utiliza la función *fin_espera()* para que se ejecute el opuesto antes de volver a parpadear.

La variable **caso2** contiene la animación opuesta por el que tiene que pasar antes de parpadear de nuevo.

- *void chequeo_tiempo_gif()*

Se llama a esta función al final del bucle principal para ver si ha pasado el tiempo necesario. El mínimo de este tiempo es el que se necesita para que se ejecute la animación, calculado haciendo la multiplicación frames x delay (número de imágenes x tiempo entre imágenes).

Una vez pasado este tiempo, si se ha recibido otro comando por puerto serie, se pasa a ejecutar este nuevo comando. En caso contrario, se le añade el tiempo extra recibido en el protocolo serie.

En ambos casos, una vez cumplido el tiempo, se pasa a leer los puertos serie 1 y 2, para obtener la respuesta recibida de las pantallas. Si la respuesta es positiva (ack) se llama a la función *condiciones*. Por otro lado, si es negativa (nack) se llama a la función *error_ack().*

- *void error_ack()*

Se recurre a ella si ha habido algún error con las pantallas en la ejecución de la animación. Imprime un mensaje en el 'hyperterminal' mostrando la respuesta obtenida. Este error suele ser debido a la disponibilidad de las pantallas y es necesario reiniciarlas.

5.4.5. FUNCIONES EN LIBRERÍA

- *int tiempo()*

Esta función actúa como temporizador. Toma de referencia el reloj interno de Arduino que cuanta en milisegundos. Se accede a él con la función *millis()*. Como se ha mencionado anteriormente, se ha establecido un tiempo de 2 segundos. Finalizado este tiempo se entiende que se ha terminado de recibir la trama de comandos. El tiempo es tan elevado puesto que si no se usa una aplicación para introducirlos, los valores se deben meter manualmente por teclado. Si se usa una aplicación, este tiempo se vería reducido. En caso de que sea necesario modificarlo, se puede acceder a él a través de la librería available.h.

- *int puerto_serie_0()*

Se trata de una de las funciones más importantes, pues es la que lee el puerto serie 0 y va almacenando los comando recibidos para su posterior procesamiento. Es aquí donde se pone a cero el temporizador cada vez que llega un nuevo comando. Este temporizador se chequea en la función anterior, *tiempo()*.

- *int puerto_serie_1()*

Esta función se encarga de leer el puerto serie 1. Esta lectura corresponde a la respuesta de la pantalla para saber si se ha ejecutado correctamente o no el gif.

- *int puerto_serie_2()*

Realiza la misma función que puerto_serie_1 pero en este caso la respuesta proviene de la pantalla derecha.

- *void procesar(int *mrepe,int *mgif,int *mojo,int *mt, int*mD)*

Una vez recibida la trama de comandos por el puerto serie 0, en esta función se procesa. Lo primero que se mira es que el comando esté completo y se trata de una trama para la función de los displays. Después a cada variable se le da su valor correspondiente, que es pasado al programa principal a través de punteros.

Estos punteros son:

- mrepe: corresponde al status.
- mgif: número del gif que se debe ejecutar.
- mojo: pantalla o pantallas que deben ejecutar el gif.
- mt: tiempo extra que se añade tras ejecutarse el gif.
- mD: es el comando 0 y debe ser la 'D' (función display).

- *void error()*

Se llama a esta función en el caso de que después de procesar el comando recibido por puerto serie de error, es decir, que no se ha llevado a cabo bien el protocolo.

La función resetea la variable en la que se guardan los comando recibidos.

- *OLED_Init(e_port puerto)*

Esta función pertenece a la librería propia de las pantallas y se usa para iniciarlas. El argumento ***puerto*** hace referencia a la pantalla que se ha de inicializar.

- *OLED_Clear(e_port puerto)*

Limpia la pantalla ordenada según el argumento ***puerto***.

5.4.6. FUNCIONES PROPIAS DE ARDUINO

El lenguaje Arduino tiene varias funciones básicas para programar. En todas las que intervenga el puerto serie, existe una modificación para especificar cuál de ellos se quiere usar. Esta modificación consiste en poner el número del puerto serie detrás de la palabra reservada **Serial**. Si el puerto serie es el 0, no hace falta poner ningún número. En la implementación de este proyecto se utilizan las siguientes funciones:

- *Void setup()*

Dentro de esta función se hacen las inicializaciones. Al estar fuera del bucle, solo se ejecuta una vez. Aunque no se vayan a realizar inicializaciones, esta función debe existir, aun estando vacía.

- *Serial.begin()*

Se usa para establecer la velocidad en baudios de los puertos serie. En este caso sería para el puerto serie 0. Si se quiere establecer la velocidad del puerto serie 1, sufriría una modificación quedando de la manera *Serial1.begin().*

- *Serial.println()*

Sirve para imprimir por pantalla un mensaje, añadiendo al final de este un retorno de carro, es decir, que lo siguiente que se escriba aparecerá en un nueva línea.

- *Void loop()*

Señala el inicio del bucle del programa principal. Solo puede existir uno por programa. Junto con la función *void setup()*, es imprescindible en un programa. Obligatoriamente debe existir, si no da fallo al compilar.

- *Millis()*

Esta función se usa en el momento de comprobar el paso del tiempo, puesto que conserva el tiempo pasado desde el inicio del programa en milisegundos. Estos milisegundos se guardan en una variable de tipo *unsigned long*, por lo que tarda en desbordarse (volver a cero) aproximadamente 50 días (2^32-1). Cada vez que se ejecuta el programa esta variable se resetea y empieza a contar de cero, por lo que hay pocas posibilidades de coincidir con su desbordamiento.

El tiempo final de los temporizadores será el almacenado en *millis()* en el momento en el que se inician más el tiempo que tiene que pasar. Así, si cuando se inicializa el temporizador, en *millis()* está almacenado el valor 32.000, si tienen que pasar dos segundos más (2000 ms), el temporizador llegará a su final cuando en *millis()* esté almacenado el valor 34.000.

Para saber que este final ha llegado se comparan los valores de configuración del temporizador (tiempo final) con el tiempo en cada instante en el que se compara.

5.4.7. APLICACIÓN CLIENTE

Como se ha mencionado anteriormente, para enviar por puerto serie del ordenador al Arduino se ha implementado una aplicación con el fin de tener una batería de posibilidades de comandos y elegir cuál se quiere mandar. Así el envío de la trama se hace más rápido y con menos posibilidades de error, de mandar un comando incorrecto.

Esta aplicación ha sido implementada en Linux, pero la trama de protocolo serie creada se puede utilizar también con el hyperterminal de Windows, en caso de no disponer de esta aplicación cliente.

Cuando se ejecuta esta aplicación lo primero que aparece es un pequeño menú (ver Figura 66) donde aparecen los comandos disponibles.

```
Introduzca numero del comando.

Pulse 1 para parpadear

Pulse 2 para reir

Pulse 3 para sospechar por pantalla izquierda

Pulse 4 para sorprenderse por pantalla derecha

Pulse 5 para permanecer triste

Pulse 6 para sospechar

Pulse 7 para sorprenderse

Pulse 8 para permanecer sonriendo

Pulse 9 para dejar de sonre
```

Figura 66: Menú de la aplicación cliente

La correspondiente trama mandada para cada uno de los comandos es la mostrada en la Tabla 7.

Número de comando	Trama mandada
1	DP00023
2	DP00101
3	DP10703
4	DP20321
5	DP00504
6	DP00702
7	DP00321
8	DP00104
9	DP00205

Tabla 7: Tramas del protocolo serie

Como se observa en la Figura 67, una vez introducido el comando deseado se muestra la elección tomada.

```
El numero introducido es: 2

Riendo
```

Figura 67: Muestra de la elección tomada

En el caso improbable de que en el código esté mal introducida la trama, al procesarlo Arduino se percata del error y manda un mensaje que es leído por la aplicación cliente y lo muestra por pantalla. Esto hace indicar que hay algún valor mal introducido, por lo que debe ser revisado. El mensaje que aparece por pantalla de la aplicación cliente se muestra en la Figura 68.

Figura 68: Mensaje mostrado

Las tramas que se han introducido han sido las que se usarán después para hacer las pruebas, pero en cualquier momento pueden ser modificadas accediendo al programa de la aplicación, cuyo nombre es “serielinux.c”, a través de un editor de texto.

Antes de lanzar la aplicación se debe comprobar el puerto serie en el que se encuentra la placa Arduino conectada al ordenador, puesto que en Linux cada vez que desconectas la placa Arduino sin reiniciar el sistema cambia la ubicación. Esto se hace para después crear un link simbólico que apunte a esta dirección. Por ejemplo, si la placa se encuentra en el puerto serie ttyACM0, se debería introducir en el terminal de comandos la siguiente instrucción: *sudo ln –s /dev/ttyACM0 /dev/ardulore*.

El link *ardulore* luego es leído en el código de la aplicación cliente.

Capítulo 6: RESULTADOS EXPERIMENTALES

6.1. INTRODUCCIÓN

Para comprobar la funcionalidad del programa se han diseñado una serie de pruebas en las que se introducen correctamente el protocolo serie, y otras en las que se introduce mal para comprobar que detecta los casos de error.

En cada ejemplo se explica lo que deben hacer las pantallas, la trama introducida y el número que hay que introducir en la aplicación cliente para que se mande dicha trama. Van acompañados de unas imágenes que hacen una idea de lo que están ejecutando las pantallas.

Cada ejemplo de protocolo correcto se corresponde con un comando del menú de la aplicación cliente. Dicho menú aparece en la Figura 69.

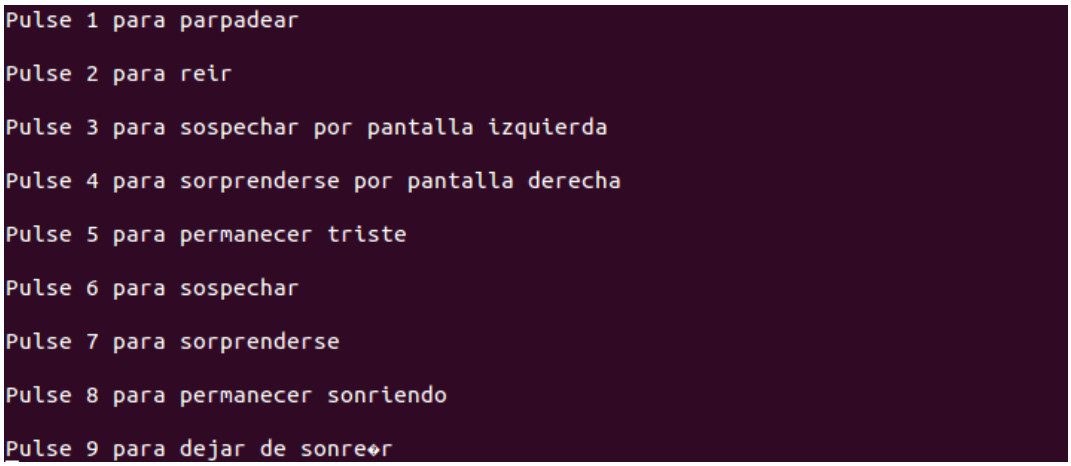

Figura 69: Menú de la aplicación cliente

6.2 PROTOCOLO CORRECTO

- **Ejemplo correcto 1**: según los valores introducidos que se muestran en la Tabla 8, debería parpadear continuamente con los 2 ojos. En la aplicación cliente se debe introducir el número 1.

BYTE	1	2	3	4	5	6	7	8
VALOR	D	P	0	0	0	2	3	INTRO

Tabla 8: Ejemplo correcto 1

La ejecución de este comando se demuestra en la Figura 70.

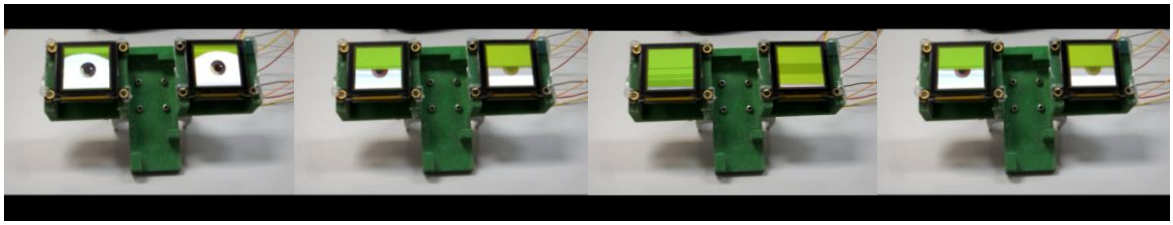

Figura 70: Parpadeo

- **Ejemplo correcto 2**: según los valores mostrados en la Tabla 9, primero debe sonreír y luego dejar de sonreír, para después parpadear. Se debe introducir el número 2 en la aplicación cliente.

BYTE	1	2	3	4	5	6	7	8
VALOR	D	P	0	0	1	0	1	INTRO

Tabla 9: Ejemplo correcto 2

La ejecución de este comando se demuestra en la Figura 71. La primera fila de imágenes corresponde al comando de sonreír. La segunda representa cuando vuelve a parpadear.

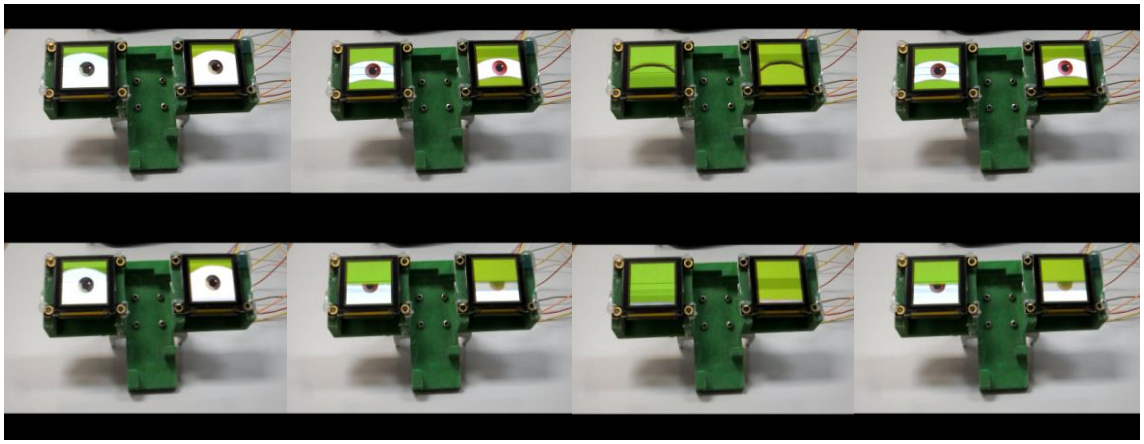

Figura 71: Sonreír y parpadeo

- **Ejemplo correcto 3**: de acuerdo con los valores que aparecen en la Tabla 10, se debe sospechar por la pantalla derecha continuamente. Se introduce el número 3.

BYTE	1	2	3	4	5	6	7	8
VALOR	D	P	1	0	7	0	3	INTRO

Tabla 10: Ejemplo correcto 3

La funcionalidad es correcta, tal como se observa en la Figura 72.

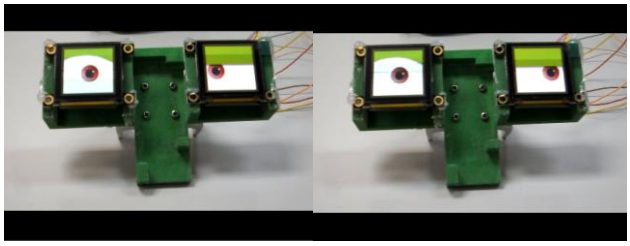

Figura 72: Sospechar pantalla derecha

- **Ejemplo correcto 4**: debe sorprenderse por el ojo izquierdo, dejar de sorprenderse, y parpadear por los 2 ojos. Corresponde al número 4 en el menú de la aplicación cliente.

BYTE	1	2	3	4	5	6	7	8
VALOR	D	P	2	0	3	2	1	INTRO

Tabla 11: Ejemplo correcto 4

La fila superior de la Figura 73 corresponde a cuando se sorprende por la pantalla izquierda, mientras que en la inferior se observa que vuelve a parpadear.

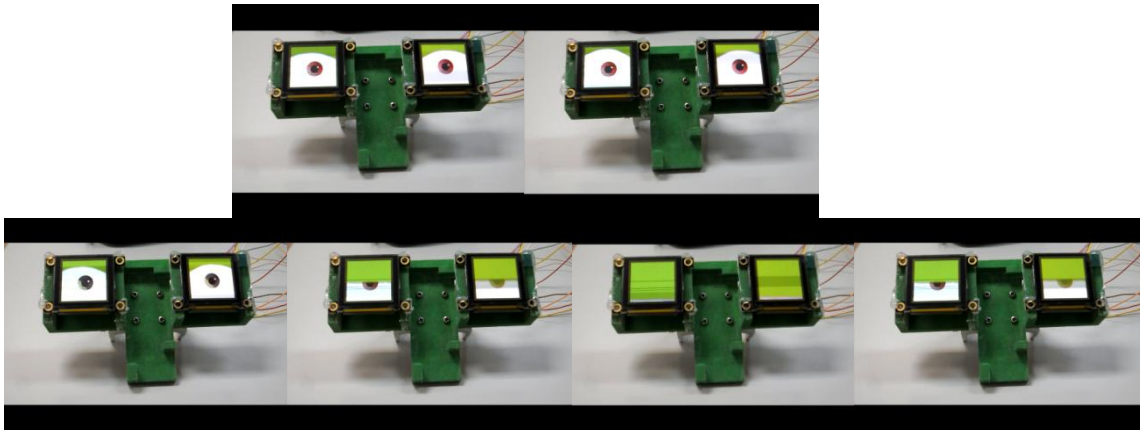

Figura 73: Sorprenderse pantalla izquierda y parpadeo

- **Ejemplo correcto 5**: como se observa en la Tabla 12, en este caso se introducen 2 tramas enteras para comprobar el paso de una a otra. Primero se debe quedar triste en los dos ojos (comando número 5), hasta que introduzcamos el siguiente comando, que será el que mande parpadear por los dos ojos continuamente (comando 1). El paso de una a otra se hará pasando el opuesto, es decir, primero dejará de estar triste y luego parpadeará.

BYTE	1	2	3	4	5	6	7	8
VALOR	D	P	0	0	5	0	4	INTRO
VALOR	D	P	0	0	0	2	3	INTRO

Tabla 12: Ejemplo correcto 5

Tal como se muestra en la Figura 74, se entristece, para dejar de estar triste cuando se ha pulsado un nuevo comando. Luego procede a parpadear.

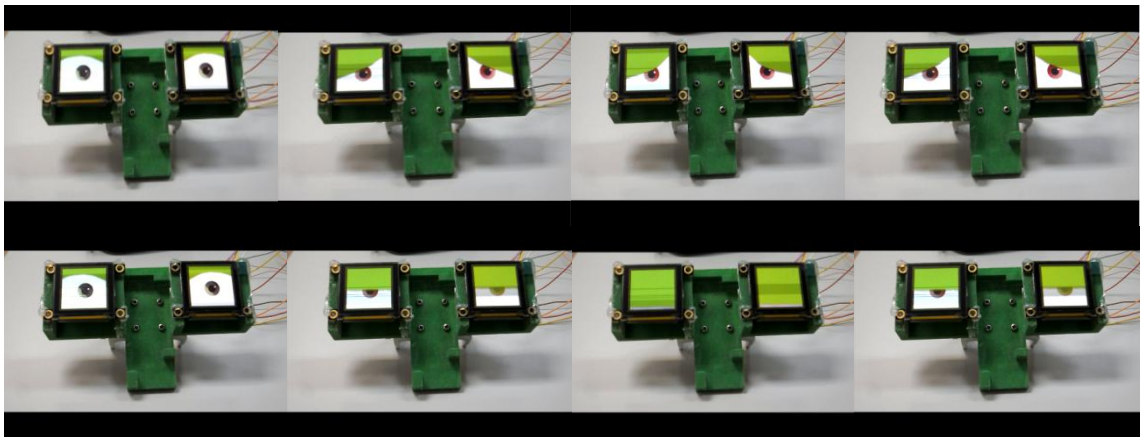

Figura 74: Permanecer triste y parpadear

- **Ejemplo correcto 6**: tal como se muestra en la Tabla 13, se tiene otra trama doble, una a continuación de la otra. Primero sospecha por los 2 ojos (comando 6), hasta que se recibe nuevo comando, que se ejecuta y pasa a parpadear. Este nuevo comando consiste en sorprenderse por los 2 ojos (comando 7) y hacer el opuesto. Tras esto, pasa a parpadear.

BYTE	1	2	3	4	5	6	7	8
VALOR	D	P	0	0	7	0	2	INTRO
VALOR	D	P	0	0	3	2	1	INTRO

Tabla 13: Ejemplo correcto 6

En la primera fila de imágenes de la Figura 75 primero sospecha y luego se sorprende. En la segunda fila, vuelve a parpadear.

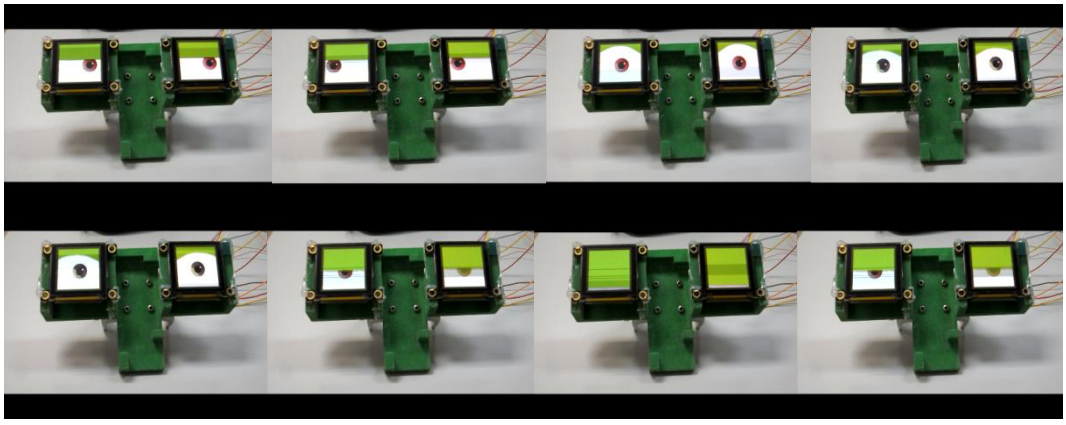

Figura 75: Sospechar, sorprenderse y parpadear

- **Ejemplo correcto 7**: esta última doble trama que aparece en la Tabla 14, consiste en primero sonreír con los dos ojos (comando 8), y mantenerse fija hasta nueva trama. Esta nueva trama consiste en dejar de sonreír (comando 9), y después pasar a parpadear por los 2 ojos.

BYTE	1	2	3	4	5	6	7	8
VALOR	D	P	0	0	1	0	4	INTRO
VALOR	D	P	0	0	2	0	5	INTRO

Tabla 14: Ejemplo correcto 7

Tal como se muestra en la Figura 76, sonríe por los dos ojos, y permanece en la posición que se muestra en la tercera imagen de la primera fila. Luego deja de sonreír y pasa a parpadear.

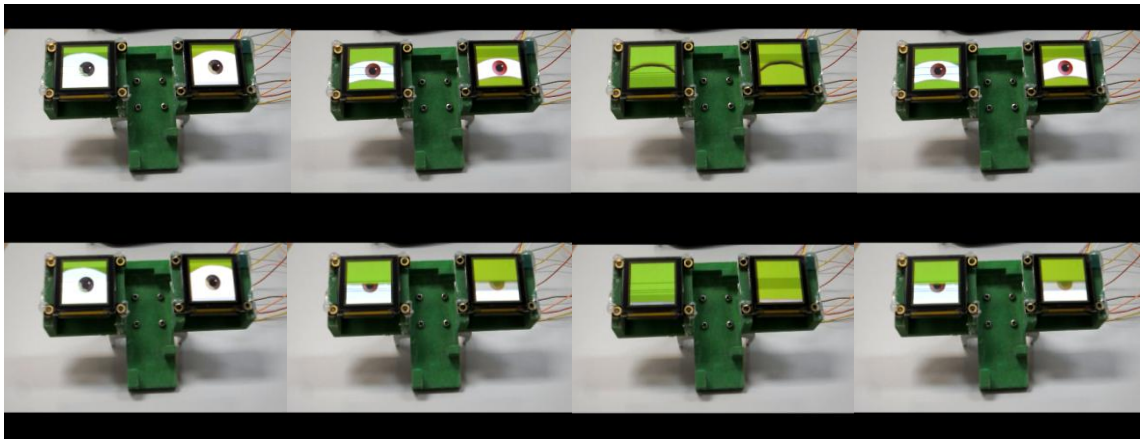

Figura 76: Permanecer riendo y parpadeo

6.3. PROTOCOLO INCORRECTO

A continuación se mostrarán tramas de protocolo serie incorrecto que hará saltar el error de comando y por lo tanto no se ejecutarán.

En todos los casos, debe salir el mensaje de error en la aplicación cliente. Si anteriormente ya se había recibido un protocolo correcto y las pantallas ya habían ejecutado alguna animación, tras la recepción de un protocolo incorrecto no se alteran, es decir, continúan con la orden anterior, que normalmente será el parpadeo.

Si es el primer protocolo que se envía cuando se ha cargado el programa en Arduino, las pantallas permanecen en negro puesto que antes de comenzar el bucle reciben la orden de limpiarse e inicializarse.

- Ejemplo incorrecto 1: no se ejecuta debido a que el valor de los ojos no es aceptado. Deber estar entre 0 y 2.

BYTE	1	2	3	4	5	6	7	8
VALOR	D	P	3	0	0	2	3	INTRO

Tabla 15: Ejemplo incorrecto 1

- Ejemplo incorrecto 2: no se ha ejecutado debido a que no se trata de la función para los displays. El inicio de la trama debe ser DP.

BYTE	1	2	3	4	5	6	7	8
VALOR	A	P	0	0	0	2	3	INTRO

Tabla 16: Ejemplo incorrecto 2

- Ejemplo incorrecto 3: no se ejecuta porque el valor de la animación no es el correcto. Debe estar comprendido entre 00 y 15.

BYTE	1	2	3	4	5	6	7	8
VALOR	D	P	0	1	6	2	3	INTRO

Tabla 17: Ejemplo incorrecto 3

- Ejemplo incorrecto 4: no se ejecuta porque la trama no está completa.

BYTE	1	2	3	4	5	6	7	8
VALOR	D	P	0	0	0	INTRO	-	-

Tabla 18: Ejemplo incorrecto 4

- Ejemplo incorrecto 5: no se ejecuta por que el valor del estado (byte 7) es superior al permitido. Debe estar entre 1 y 5.

BYTE	1	2	3	4	5	6	7	8
VALOR	D	P	0	0	0	0	6	INTRO

Tabla 19: Ejemplo incorrecto 5

En todos los ejemplos anteriores, el resultado es el mismo: mostrar el error en la aplicación cliente (ver Figura 77). Este error también se puede observar en el monitor serie propio de Arduino, puesto que imprime todos los mensajes que manda Arduino a través del puerto serie 0 (ver Figura 78).

```
El numero introducido es: 9

Dejando de sonre◆r
dato leido: error
```

Figura 77: Mensaje de error en la aplicación cliente

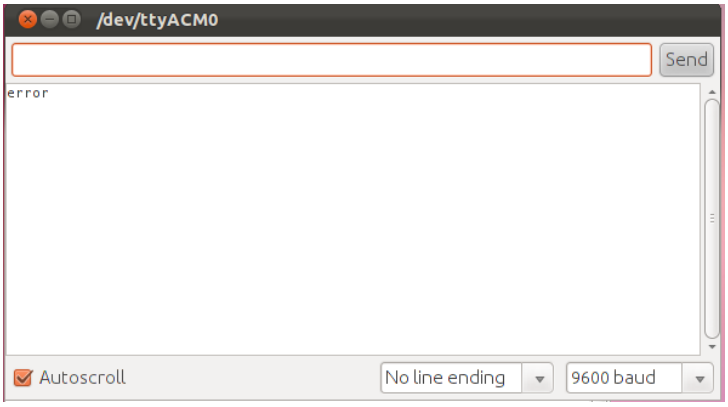

Figura 78: Mensaje de error en el monitor serie

Capítulo 7: CONCLUSIONES Y TRABAJOS FUTUROS

7.1. CONCLUSIONES

Como resultado de este PFC se ha obtenido un sistema que permite mostrar mediante pantallas expresiones básicas humanas, como sonreír o enfadarse, usando unos displays que son controlados mediante los puertos serie de Arduino, que actúa como multiplexor.

Se ha propuesto un protocolo que sirve como capa de abstracción del hardware hacia una capa con información general, en la que se habla de expresiones a alto nivel. El protocolo permite una fácil comunicación entre la aplicación cliente y Arduino, ya que todos los datos que se deben mandar por puerto serie quedan ordenados y completos. Este protocolo serie ofrece las ventajas de poder comunicar al microcontrolador Arduino lo que se quiere mostrar con los displays, como qué animación ejecutar y durante cuánto tiempo, de una manera cómoda. Además permite a Arduino conocer a qué pantalla debe enviar los comandos, permitiéndole actuar como multiplexor, puesto que de una entrada obtiene dos posibles salidas.

A parte de la creación del nuevo protocolo serie, se ha estudiado el protocolo ya existente entre Arduino y las pantallas, permitiendo comprender la completa funcionalidad de la que disponen los displays.

En lo respectivo al código implementado, se ha encapsulado parte de él en unas librerías que se añaden al programa principal, con motivo de simplificarlo. Además, este encapsulamiento permite tener recogidas las funciones de lectura de puerto serie, puesto que si se desea cambiar el protocolo serie o ampliarlo, lo primero que se deben cambiar son estas funciones. A parte de la lectura del puerto serie, también se ha realizado otra librería en la que aparecen las constantes que se usan a lo largo del proyecto, para una fácil modificación.

Por último, se ha realizado un primer cliente que ha permitido realizar las pruebas mostradas en los resultados experimentales, y que tiene comunicación serie con Arduino, puesto que envía y recibe por puerto serie.

Debido a todo lo anterior, se ha obtenido un gran conocimiento de los microcontroladores Arduino, de las pantallas uOLED, de los protocolos serie, del encapsulamiento en librerías y de la creación de aplicaciones cliente.

7.2. TRABAJOS FUTUROS

Como consecuencia del trabajo de este PFC y de las primeas pruebas realizadas, se proponen realizar:

- Completar el protocolo añadiendo nuevas expresiones y mejorando la continuidad entre expresiones.

- Dotar de más funcionalidades al Arduino. En este proyecto se usa el Arduino Mega 2560, el cual dispone de múltiples pines analógicos y digitales y de generación de PWM. Esto permite a Arduino interactuar con su entorno, que es una de las finalidades que se exigen a un robot personal y de servicios. Por esto, se propone como trabajo futuro la utilización de estos pines aprovechando que la placa Arduino estará integrada dentro del robot.

- Otro trabajo futuro sería el de sustituir la aplicación cliente por un nodo Ros (*Robot Operating System)* que permita su fácil integración dentro de una arquitectura software de un robot de alto nivel, para que gestione los displays para la expresión del robot de acuerdo a diferentes aplicaciones de interacción. Para esto también sería necesario modificar el protocolo serie.

- También se propone realizar una ampliación de la aplicación cliente. Esta ampliación consistiría en aumentar el menú disponible, es decir, dar más opciones de tramas para enviar por puerto serie. además, también se podría poder elegir la opción de introducir otros comandos distintos fuera del menú. Por último, otra ampliación posible sería la realización de más lecturas del puerto serie 0, para mostrar mensajes por la aplicación cliente.

- Como último trabajo futuro se propone establecer una conexión del hardware fija, es decir, acoplar los elementos (placa Arduino y pantallas) a un robot con el fin de que las uniones se mantengan permanentes. Para ello los cables que conectan Arduino con los displays deberían ser rehechos.

REFERENCIAS

[1] Sosnowski, S, Bittermann, A. ; Kuhnlenz, K. ; Buss, M. Design and Evaluation of Emotion-Display EDDIE. Intelligent Robots and Systems, 2006 IEEE/RSJ International. pp. 3113 - 3118. 2006.

[2] Lesley Axelrod.The affective connection: how and when users communicate emotion. Human factors in computing systems. pp. 1033-1034. 2004 Article.

[3] Hashimoto, T. , Hiramatsu, S. ; Kobayashi, H. Dynamic display of facial expressions on the face robot made by using a life mask. The 8th IEEE-RAS International Conference on Humanoids.pp. 521 - 526. 2008. Date of Conference: 1-3 Dec. 2008.

[4] http://arduino.cc/es/ , página web de Arduino en español. No todos los apartados que tiene en inglés están traducidos. Consultada por última vez el día 26/06/12.

[5] http://arduino.cc/forum/index.php?board=32.0, foro de la página de Arduino en español. Consultada por última vez el día 26/06/12.

[6] http://arduino.cc , página web de Arduino en inglés. Está más actualizada. Consultada por última vez el día 26/06/12.

[7] http://arduino.cc/es/Main/Software, desde aquí se puede descargar el software para empezar a utilizar Arduino. Consultada por última vez el día 29/06/12.

[8] Libro “Arduino programming notebook”. Autor Brian W. Evans. Editorial “Creative Commons”. Segunda edición, 2008.

[9]http://arduino.cc/es/Tutorial/HomePage, librerías compatibles con Arduino. Consultada por última vez el día 29/06/12.

[10] http://iteadstudio.com, consultada por última vez el día 28/06/12.

[11]http://iteadstudio.com/store/images/produce/Display/TFT/TFTLCD_2.2SPI/HX8340-B_N__DS_preliminary_v01_071203.pdf, hoja de características. Consultada por última vez el día 28/06/12.

[12]http://iteadstudio.com/store/images/produce/Display/TFT/TFTLCD_ITDB02_2.4/DS_S6D1121.pdf, descarga de hoja de características. Consultada por última vez el día 28/06/12.

[13]http://www.4dsystems.com.au/prod.php?id=125 . Consultada por última vez el día 26/06/12.

[14]http://a.parsons.edu/~turkb970/blog/?page_id=66 . Consultada por última vez el día 26/06/12.

[15]http://www.4dsystems.com.au/downloads/Serial-Display-Modules/uOLED-128-G1%28SGC%29/Docs/uOLED-128-G1SGC-DS-rev6.pdf . Descarga de manual de instrucciones de las pantallas. Consultada por última vez el día 26/06/12.

[16] Proyecto "**Diseño, desarrollo y manual de una interfaz gráfica distribuida para Maggie**". Realizado por Laura Romero Bachiller, en Octubre del 2010.

[17]http://www.4dsystems.com.au/downloads/Semiconductors/GOLDELOX-SGC/Docs/GOLDELOX-SGC-COMMANDS-SIS-rev6.pdf. Descarga del manual de protocolo serie de las pantallas. Consultada por última vez el día 26/12/06.

[18]http://code.google.com/p/displayshield4d/downloads/list. Descarga de la librería. También se puede encontrar el código con un ejemplo de uso de la librería. Consultada por última vez el día 26/06/12.

[19]http://blog.bricogeek.com/noticias/arduino/libreria-4display-shield-para-arduino/. Consultada por última vez el día 27/06/12.

ANEXOS

A.1.CÓDIGO PRINCIPAL

```
///////declaración de funciones//////////////////////////////////////////////////////////////

void condiciones(int rep,int caso,int caso2,int opo);
void comandos_gif(char del,char nframes,long address, int ojos);
void fin_espera (int caso2);
void chequeo_tiempo_gif();
void error_ack();

////////librerías/////////////////////////////////////////////////////////////////////////////

#include <valores.h>
#include <_oled160drv.h>
#include <available_simpli.h>
Available simpli;   //instanciar la clase

////////declaración de variables//////////////////////////////////////////////////////////////

char comando[13],*mensaje;
unsigned long tiempoINT,tiempoFIN;
int i=0,j=0,ida_tiempo=0,rep4,tiempo_gif,opuesto,tiempoEXTRA,flag;
int repetir,ida_opuesto,siempre,miojo=0,mit,mirepe,ack1=0,ack2=0,caso,caso2,ojo;
int gif=0,D,P,CR,comando_recibido,recibido,migif,miD,_error;

////////////////////////////////////////////////////////////////////////////////////////////////

void setup(){

  //inicialización de los puertos serie

  Serial.begin (9600);
  Serial1.begin(9600);
  Serial2.begin(9600);

  //inicialización de las pantallas

  OLED_Init(PORT_1);
   OLED_Clear(PORT_1);
   OLED_Init(PORT_2);
   OLED_Clear(PORT_2);

}
```

```
////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////
void loop(){

 //lectura del puerto serie

 flag= simpli.puerto_serie_0();

 if (flag==1){

  simpli.procesar(&mirepe,&migif,&miojo,&mit,&miD);
  flag=0;
  comando_recibido=1;
  recibido=1;
     }

 //fin serial available

  if (miD==68)//&(P==80)&(CR==13)) //comprueba que la funcion es para los displays y que hay retorno de carro

    {
     if (ida_tiempo==0)
     {
      if (comando_recibido==1)
      {
       gif=migif;
       repetir=mirepe;
       comando_recibido=0;
       tiempoEXTRA=mit;
       D=miD;
       ojo=miojo;
      }


     switch (gif){

     case 0:
      //parpadeo
      if (rep4==1){
       rep4=0;
       fin_espera(opuesto);
      }
      else {
      repetir=3;
      ojo=0;
      caso=_parpadeo;
      caso2=_normal;
      tiempo_gif=TiempoParpadeo;
```

```
  mensaje=*mensaje_parpadeo;
  comandos_gif(parpadeo[0],parpadeo[1],parpadeo[2],ojo);
  }

break;


case 1:
  //riendose
  caso=_riendose;
  caso2=_noriendose;
  tiempo_gif=TiempoRiendose;
  mensaje=*mensaje_riendose;
  comandos_gif(riendose[0],riendose[1],riendose[2],ojo);
break;


case 2:
  //noriendose
  caso=_noriendose;
  caso2=_riendose;
  mensaje=*mensaje_noriendose;
  tiempo_gif=TiempoNoriendose;
  comandos_gif(noriendose[0],noriendose[1],noriendose[2],ojo);
break;


case 3:
  //sorprendido
  caso=_sorprendido;
  caso2=_nosorprendido;
  mensaje=*mensaje_sorprendido;
  tiempo_gif=TiempoSorprendido;
  comandos_gif(sorprendido[0],sorprendido[1],sorprendido[2],ojo);
break;


case 4:
  //no_sorprendido
  caso=_nosorprendido;
  caso2=_sorprendido;
  mensaje=*mensaje_nosorprendido;
  tiempo_gif=TiempoNosorprendido;
  comandos_gif(nosorprendido[0],nosorprendido[1],nosorprendido[2],ojo);
break;

case 5:
  //triste
  caso=_triste;
  caso2=_notriste;
```

```
  mensaje=*mensaje_triste;
  tiempo_gif=TiempoTriste;
  comandos_gif(triste[0],triste[1],triste[2],ojo);
break;


case 6:
  //no_triste
   caso=_notriste;
  caso2=_triste;
  mensaje=*mensaje_notriste;
  tiempo_gif=TiempoNotriste;
  comandos_gif(notriste[0],notriste[1],notriste[2],ojo);
break;


case 7:
  //sospechoso
  caso=_sospechoso;
  caso2=_normal;
  mensaje=*mensaje_sospechoso;
  tiempo_gif=TiempoSospechoso;
  comandos_gif(sospechoso[0],sospechoso[1],sospechoso[2],ojo);
break;


case 8:
  //normal
  caso=_normal;
  caso2=_normal;
  mensaje=*mensaje_normal;
  tiempo_gif=TiempoNormal;
  comandos_gif(normal[0],normal[1],normal[2],ojo);
break;


case 9:
  //enfadado_izquierdo
  caso=_enfadandose;
  caso2=_noenfadandose;
  mensaje=*mensaje_enfadado;
  tiempo_gif=TiempoEnfadandose;
  comandos_gif(enfadandose[0],enfadandose[1],enfadandose[2],ojo);
break;


case 10:
  //noenfadandose
  caso=_noenfadandose;
  caso2=_enfadandose;
```

```
  mensaje=*mensaje_noenfadado;
  tiempo_gif=TiempoNoenfadandose;
  comandos_gif(noenfadandose[0],noenfadandose[1],noenfadandose[2],ojo);
break;


case 11:
  //abriendo
  caso=_abriendo;
  caso2=_cerrando;
  mensaje=*mensaje_abriendo;
  tiempo_gif=TiempoAbriendob;
  comandos_gif(abriendob[0],abriendob[1],abriendob[2],ojo);
break;


case 12:
  //cerrando
  caso=_cerrando;
  caso2=_abriendo;
  mensaje=*mensaje_cerrando;
  tiempo_gif=TiempoCerrandob;
  comandos_gif(cerrandob[0],cerrandob[1],cerrandob[2],ojo);
break;


case 13:
  //abriendo
  caso=_abriendob;
  caso2=_cerrandob;
  mensaje=*mensaje_abriendo;
  tiempo_gif=TiempoAbriendo;
  comandos_gif(abriendo[0],abriendo[1],abriendo[2],ojo);
break;


case 14:
   //cerrando
  caso=_cerrandob;
  caso2=_abriendob;
  mensaje=*mensaje_cerrando;
  tiempo_gif=TiempoCerrando;
  comandos_gif(cerrando[0],cerrando[1],cerrando[2],ojo);
break;


}//fin del switch
 }//fin del ida_tiempo
else
{
```

```
      chequeo_tiempo_gif();
     } //funcion que chequea si ha pasado el tiempo necesario para que se ejecute el gif


  }//fin del if DP+CR


} //fin del loop

////////////////////////////////////////////////////////////////////////////////////////////////////
////
//inicio funciones
////////////////////////////////////////////////////////////////////////////////////////////////////
////

void condiciones(int rep,int caso,int caso2,int opo){
  ida_tiempo=0;

        if ((rep==1)&(opo==1))
        {
          ida_opuesto=0;
          gif=_parpadeo;
        }
        else if (rep==1)//opuesto
        {
          gif=caso2;
          ida_opuesto=1;
        }
        else if (rep==2)//repetir hasta nuevo comando
        {
          gif=caso;
        }
        else if (rep==3)//repetir siempre
        {
          siempre=caso;
        }
        else if (rep==4)//esperar hasta nuevo comando
        {
          rep4=1;
          opuesto=caso2;
          miD=0;
        }
        else {

          opuesto=0;
          gif=siempre;
       }
        }

////////////////////////////////////////////////////////////////////////////////////////////////
```

```
void comandos_gif(char del,char nframes,long address, int ojos)
{
 char framesmid;
 char frameslo;
 char addhi;
 char addmid;
 char addlo;
 addlo=(address)&(0x0000FF);
 addmid=((address)&(0x00FF00))>>8;
 addhi=((address)&(0xFF0000))>>16;
 frameslo=(nframes)&(0x00FF);
 framesmid=((nframes)&(0xFF00))>>8;
 if (ojos==1){
  Serial1.print(0x40,BYTE);//@
  Serial1.print(0x56,BYTE);//v
  Serial1.print(0x00,BYTE);//x
  Serial1.print(0x00,BYTE);//y
  Serial1.print(0x80,BYTE);//width
  Serial1.print(0x80,BYTE);//heigh
  Serial1.print(0x10,BYTE);//colourmode
  Serial1.print(del,BYTE);//delay
  Serial1.print(framesmid,BYTE);//frames(msb)
  Serial1.print(frameslo,BYTE);//frames(lsb)
  Serial1.print(addhi,BYTE);//sectorad(hi)
  Serial1.print(addmid,BYTE);//sectorad(mid)
  Serial1.print(addlo,BYTE);//sectorad(lo)
 }
 else if (ojos==2)
 {
  Serial2.print(0x40,BYTE);//@
  Serial2.print(0x56,BYTE);//v
  Serial2.print(0x00,BYTE);//x
  Serial2.print(0x00,BYTE);//y
  Serial2.print(0x80,BYTE);//width
  Serial2.print(0x80,BYTE);//heigh
  Serial2.print(0x10,BYTE);//colourmode
  Serial2.print(del,BYTE);//delay
  Serial2.print(framesmid,BYTE);//frames(msb)
  Serial2.print(frameslo,BYTE);//frames(lsb)
  Serial2.print(addhi,BYTE);//sectorad(hi)
  Serial2.print(addmid,BYTE);//sectorad(mid)
  Serial2.print(addlo,BYTE);//sectorad(lo)
 }
 else if (ojos==0)
 {
  Serial1.print(0x40,BYTE);//@
  Serial1.print(0x56,BYTE);//v
  Serial1.print(0x00,BYTE);//x
  Serial1.print(0x00,BYTE);//y
```

```
  Serial1.print(0x80,BYTE);//width
  Serial1.print(0x80,BYTE);//heigh
  Serial1.print(0x10,BYTE);//colourmode
  Serial1.print(del,BYTE);//delay
  Serial1.print(framesmid,BYTE);//frames(msb)
  Serial1.print(frameslo,BYTE);//frames(lsb)
  Serial1.print(addhi,BYTE);//sectorad(hi)
  Serial1.print(addmid,BYTE);//sectorad(mid)
  Serial1.print(addlo,BYTE);//sectorad(lo)

  Serial2.print(0x40,BYTE);//@
  Serial2.print(0x56,BYTE);//v
  Serial2.print(0x00,BYTE);//x
  Serial2.print(0x00,BYTE);//y
  Serial2.print(0x80,BYTE);//width
  Serial2.print(0x80,BYTE);//heigh
  Serial2.print(0x10,BYTE);//colourmode
  Serial2.print(del,BYTE);//delay
  Serial2.print(framesmid,BYTE);//frames(msb)
  Serial2.print(frameslo,BYTE);//frames(lsb)
  Serial2.print(addhi,BYTE);//sectorad(hi)
  Serial2.print(addmid,BYTE);//sectorad(mid)
  Serial2.print(addlo,BYTE);//sectorad(lo)
 }
 tiempoINT=millis();
 tiempoFIN=tiempo_gif+tiempoINT+tiempoEXTRA;
 ida_tiempo=1;
 //Serial.println(mensaje);
 recibido=0;
}
////////////////////////////////////////////////////////////////////////////////////////////////

void fin_espera (int caso2)
{
 gif=caso2;
 siempre=_parpadeo;
 repetir=0;
}

////////////////////////////////////////////////////////////////////////////////////////////////

void chequeo_tiempo_gif(){

 if (((tiempoFIN-tiempoEXTRA)<millis())&(recibido==1))
     {
       ack1= simpli.puerto_serie_1();
       ack2= simpli.puerto_serie_2();
      if (((ojo==1)&&(ack1==6))||((ojo==2)&&(ack2==6))||((ojo==0)&(ack1==6)&(ack2==6)))

       {
```

```
        condiciones(repetir,caso,caso2,ida_opuesto);
        }
        else
        {
          error_ack();
        }
      }
 else
 if (tiempoFIN<millis())
      {
         ack1= simpli.puerto_serie_1();
         ack2= simpli.puerto_serie_2();
       if (((ojo==1)&&(ack1==6))||((ojo==2)&&(ack2==6))||((ojo==0)&&(ack1==6)&&(ack2==6)))
        {
        condiciones(repetir,caso,caso2,ida_opuesto);
        }
        else
        {
          error_ack();
        }

      }
}

////////////////////////////////////////////////////////////////////////////////////////

void error_ack(){
  Serial.print("ack1=");
  Serial.println(ack1);
  Serial.print("ack2=");
  Serial.println(ack2);
  ida_tiempo=0;
}
```

A.2.LIBRERÍA AVAILABLE.H

Aquí encontramos dos ficheros, uno el propio de Available.h, donde están declaradas las funciones, y otro el de Available.cpp, donde están implementadas las funciones.

A.2.1. Available.h

```
#ifndef Available_h
#define Available_h
#include <WProgram.h>


#define t_e 2000;

////////////////////////////////////////////////////////////////
//definicion de la clase


class Available
{

 public:

   //constructor

   Available();

   //funciones

   int tiempo();
   int puerto_serie_0();
   int puerto_serie_1();
   int puerto_serie_2();
   void procesar(int *mrepe,int *mgif,int *mojo,int *mt, int*mD);
   void error();

   //variables

   int tiempo_espera_comando;


 private:

   //variables

   int com,_flag,__flag,i;
   char comando[13];
```

```
   unsigned long temporizador,temporizador_fin;

};

#endif
```

A.2.2. Available.cpp

```
#include "available_simpli.h"



////////////////////////////////////////////////////////////////////////////////////////////



/*funciones que se pueden encontrar:
        tiempo(): temporizador para saber si se ha terminado de mandar un comando
        puerto_serie_0: lee los comandos que van llegando al puerto serie 0
        procesar: procesa los comandos recibidos
        puerto_serie_1: lee el puerto serie 1
        puerto_serie_2: lee el puerto serie 2
*/

////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////




Available::Available()
{
 tiempo_espera_comando=t_e;
}


////////////////////////////////////////////////////////////////////////////////////////////


int Available::tiempo()
{
 if (temporizador_fin<millis())
 {
   com=0;
   _flag=1;
   i=0;
 }
```

```
 return _flag;
}


//////////////////////////////////////////puerto serie 0////////////////////////////////


int Available::puerto_serie_0 ()
{
 if (Serial.available()>0)
 {
  comando[i]=Serial.read();
  /*Serial.print("el comando ");
  Serial.print(i);
  Serial.print(" es:");
  Serial.println(comando[i]);*/
  i++;
  com=1;
  _flag=0;
  temporizador=millis();
  temporizador_fin=temporizador+tiempo_espera_comando;
 }
 else if (com==1)
 {
  __flag=tiempo();
 }
 else {__flag=0;}
 return __flag;
}



//////////////////////////////////////////////////////////////////////////////////////////

void Available::procesar(int *repe,int *gif,int *ojo,int *t,int *D)
{

 if ((comando[0]==68)&(comando[7]==13))
  {
   if          ((((comando[6]-48)>0)&((comando[6]-48)<6))&(((comando[3]-48)*10+comando[4]-
48)<16)&((comando[2]-48)<3))
   {
    *repe=comando[6]-48;
    *gif=(comando[3]-48)*10+comando[4]-48;
    *ojo=comando[2]-48;
    *t=(comando[5]-48)*1000;
    *D=comando[0];
    comando[0,1,2,3,4,5,6,7]=0;
   }
   else { error();}
```

```
  }
 else
 {
   error();
 }
}


//////////////////////////////////////puerto serie 1////////////////////////////////

int Available::puerto_serie_1()
{
 int _ack1;
 if (Serial1.available()>0)
 {
   _ack1=Serial1.read();
 }
 return _ack1;
}


//////////////////////////////////////puerto serie 2////////////////////////////////

int Available::puerto_serie_2()
{
 int _ack2=0;
 if (Serial2.available()>0)
 {
   _ack2=Serial2.read();
 }
 return _ack2;
}

//////////////////////////////////////////error/////////////////////////////////////

void Available::error()
{
 Serial.println("error");
 i=0;

 comando[0,1,2,3,4,5,6,7]=0;
 }
```

A.3.LIBRERÍA _OLED160

A parte de la librería utilizada finalmente en el proyecto, se acompaña de la original, llamada oled160

A.1.1. _oled160.h (final)

```
/*********************************************************

 uOLED-160-GMD1 Driver for Arduino
 Code: Oscar Gonzalez
 August 2007

 www.bricogeek.com

Modified by:Burcum Turkmen
May 2010

www.burcumturkmen.com
 *********************************************************/

#include <stdint.h>
#include <WProgram.h>
#include <stdio.h>

#define OLED_BAUDRATE                   9600//57600
#define OLED_RESETPIN                   8     // PIN of reset
#define OLED_INITDELAYMS                5000

// Initialisation routine
#define OLED_DETECT_BAUDRATE                        0x55

// Drawing routines
#define OLED_CLEAR                          0x45
#define OLED_BKGCOLOR                       0x42
#define OLED_COPYPASTE                      0x63

#define OLED_LINE                           0x4C
#define OLED_CIRCLE                         0x43
#define OLED_CIRCLEFILL                     0x69
#define OLED_PUTPIXEL                       0x50
#define OLED_READPIXEL                      0x52
#define OLED_RECTANGLE                      0x72
#define OLED_PAINTAREA                      0x70
```

```
// Text properties
#define OLED_SETFONTSIZE                0x46
  #define       OLED_FONT5X7                    0x01
  #define       OLED_FONT8X8                    0x02
  #define       OLED_FONT8X12                   0x03
#define OLED_TEXTFORMATED               0x54

// OLED Control
#define OLED_COMMAND_CONTROL                    0x59
#define OLED_COMMAND_DISPLAY                    0x01
#define OLED_COMMAND_CONTRAST                   0x02
#define OLED_COMMAND_POWER                      0x03

#define OLED_ACK  0x06  // Ok
#define OLED_NAK  0x15  // Error


typedef enum
{
PORT_1 = 1,
PORT_2 = 2
} e_port;

void OLED_ResetDisplay()
{
  digitalWrite(OLED_RESETPIN, 0);
  delay(20);
  digitalWrite(OLED_RESETPIN, 1);
  delay(20);

}

char OLED_GetResponse(e_port puerto)
{
  byte incomingByte = OLED_ACK;

  // Wait for data available
if (puerto==PORT_1){
  while (!Serial1.available()) { delay(1); }

  // Read incoming byte
  incomingByte = Serial1.read();

}
else if (puerto==PORT_2){
while (!Serial2.available()) { delay(1); }

  incomingByte = Serial2.read();

}
```

```
return incomingByte;

}

// Initialise OLED display. You must first activate a serial comunication!
void OLED_Init(e_port puerto)
{
  // First reset display
  OLED_ResetDisplay();

  delay(OLED_INITDELAYMS);                  // Wait for init
  // Autodetect baudrate
if (puerto==PORT_1){
  Serial1.print(OLED_DETECT_BAUDRATE, BYTE);
  OLED_GetResponse(PORT_1);
}
else if (puerto=PORT_2){
  Serial2.print(OLED_DETECT_BAUDRATE, BYTE);

  OLED_GetResponse(PORT_2);
}

}

// Get 16bits value from RGB (0 to 63, 565 format)
int GetRGB(int red, int green, int blue)
{

        char text[255];

        int outR = ((red * 31) / 255);
        int outG = ((green * 63) / 255);
        int outB = ((blue * 31) / 255);

        return (outR << 11) | (outG << 5) | outB;
}

void OLED_Clear(e_port puerto)
{
if (puerto==PORT_1){
  Serial1.print(OLED_CLEAR, BYTE);
//printByte(OLED_CLEAR); // Pixel write
delay(20);
    OLED_GetResponse(PORT_1);
}
else if (puerto==PORT_2){
Serial2.print(OLED_CLEAR, BYTE);
//printByte(OLED_CLEAR); // Pixel write
delay(20);
    OLED_GetResponse(PORT_2);
```

```
}

}

void OLED_PutPixel(char x, char y, int color, e_port puerto)
{

        // Putpixel
if (puerto==PORT_1){
  Serial1.print(OLED_PUTPIXEL, BYTE);
  Serial1.print(x, BYTE);
  Serial1.print(y, BYTE);

        // Color
  Serial1.print(color >> 8, BYTE);
  Serial1.print(color & 0xFF, BYTE);        //LBS
OLED_GetResponse(PORT_1);
}
else if (puerto==PORT_2){
  Serial2.print(OLED_PUTPIXEL, BYTE);
  Serial2.print(x, BYTE);
  Serial2.print(y, BYTE);
  Serial2.print(color >> 8, BYTE);
  Serial2.print(color & 0xFF, BYTE);
OLED_GetResponse(PORT_2);
}




}

void OLED_DrawLine(char x1, char y1, char x2, char y2, int color, e_port puerto)
{

        // Line drawing
if (puerto==PORT_1){
  Serial1.print(OLED_LINE, BYTE);
  Serial1.print(x1, BYTE);
  Serial1.print(y1, BYTE);
  Serial1.print(x2, BYTE);
  Serial1.print(y2, BYTE);

        // Color
  Serial1.print(color >> 8, BYTE);
  Serial1.print(color & 0xFF, BYTE);        // MSB
OLED_GetResponse(PORT_1);
}
```

```
else if (puerto==PORT_2){
  Serial2.print(OLED_LINE, BYTE);
  Serial2.print(x1, BYTE);
  Serial2.print(y1, BYTE);
  Serial2.print(x2, BYTE);
  Serial2.print(y2, BYTE);
  Serial2.print(color >> 8, BYTE);
  Serial2.print(color & 0xFF, BYTE);
OLED_GetResponse(PORT_2);
}


}

void OLED_DrawRectangle(char x, char y, char width, char height, char filled, int color, e_port
puerto)
{

        //int color = 0xFFFF;

        // Rectangle drawing
if (puerto==PORT_1){
  Serial1.print(OLED_RECTANGLE, BYTE);
  Serial1.print(x, BYTE);
  Serial1.print(y, BYTE);
  Serial1.print(x+width, BYTE);
  Serial1.print(y+height, BYTE);

        // Color
  Serial1.print(color >> 8, BYTE);
  Serial1.print(color & 0xFF, BYTE);

        /*
        if (filled == 1) { printByte(0x01); }          // Filled
        else { printByte(0x00); }                              // Outline
        */
OLED_GetResponse(PORT_1);
}
else if (puerto==PORT_2){
  Serial2.print(OLED_RECTANGLE, BYTE);
  Serial2.print(x, BYTE);
  Serial2.print(y, BYTE);
  Serial2.print(x+width, BYTE);
  Serial2.print(y+height, BYTE);
  Serial2.print(color >> 8, BYTE);
  Serial2.print(color & 0xFF, BYTE);
OLED_GetResponse(PORT_2);

}
```

```
}

void OLED_DrawCircle(char x, char y, char radius, char filled, int color, e_port puerto)
{
if (puerto==PORT_1){
  Serial1.print(OLED_CIRCLE, BYTE);
  Serial1.print(x, BYTE);
  Serial1.print(y, BYTE);
  Serial1.print(radius, BYTE);

            // Color
  Serial1.print(color >> 8, BYTE);
  Serial1.print(color & 0xFF, BYTE);

       OLED_GetResponse(PORT_1);
}
else if (puerto==PORT_2){
  Serial2.print(OLED_CIRCLE, BYTE);
  Serial2.print(x, BYTE);
  Serial2.print(y, BYTE);
  Serial2.print(radius, BYTE);
  Serial2.print(color >> 8, BYTE);
  Serial2.print(color & 0xFF, BYTE);

       OLED_GetResponse(PORT_2);
}



}

/*
 Change font format:
 FontType can be:
    OLED_FONT5X7
    OLED_FONT8X8
    OLED_FONT8X12
*/
void OLED_SetFontSize(char FontType,e_port puerto)

{
if (puerto==PORT_1){
    Serial1.print(OLED_SETFONTSIZE, BYTE);
    Serial1.print(FontType, BYTE);

    OLED_GetResponse(PORT_1);
}
else if (puerto==PORT_2){
Serial2.print(OLED_SETFONTSIZE, BYTE);
    Serial2.print(FontType, BYTE);
```

```
   OLED_GetResponse(PORT_2);
}
}

void OLED_DrawText(char column, char row, char font_size, char *mytext, int color, e_port
puerto)
{
        //char mytext[] = "Arkanoid by pK";
if (puerto==PORT_1){
Serial1.print(0x73, BYTE);


     // Adjust to center of the screen (26 Columns at font size 0)
     int newCol = 13 - (strlen(mytext)/2);

     Serial1.print(column, BYTE);
     Serial1.print(row, BYTE);
Serial1.print(font_size, BYTE);
        //printByte(font_size); // font size (0 = 5x7 font, 1 = 8x8 font, 2 = 8x12 font)

        // Color
Serial1.print(color >> 8, BYTE);
Serial1.print(color & 0xFF, BYTE);


        for (int i=0 ; i<strlen(mytext) ; i++)
        {
 Serial1.print(mytext[i], BYTE);

        }
Serial1.print(0x00, BYTE);


     OLED_GetResponse(PORT_1);

}
else if (puerto==PORT_2){
Serial2.print(0x73, BYTE);


     // Adjust to center of the screen (26 Columns at font size 0)
     int newCol = 13 - (strlen(mytext)/2);

     Serial2.print(column, BYTE);
     Serial2.print(row, BYTE);
Serial2.print(font_size, BYTE);
        //printByte(font_size); // font size (0 = 5x7 font, 1 = 8x8 font, 2 = 8x12 font)

        // Color
```

```
Serial2.print(color >> 8, BYTE);
Serial2.print(color & 0xFF, BYTE);


        for (int i=0 ; i<strlen(mytext) ; i++)
        {
 Serial2.print(mytext[i], BYTE);

        }
Serial2.print(0x00, BYTE);


    OLED_GetResponse(PORT_2);
}
}

void OLED_DrawSingleChar(char column, char row, char font_size, char MyChar, int color,
e_port puerto)
{
if (puerto==PORT_1){
    Serial1.print(OLED_TEXTFORMATED, BYTE);

        Serial1.print(MyChar, BYTE);
    Serial1.print(column, BYTE);
    Serial1.print(row, BYTE);


        // Color
Serial1.print(color >> 8, BYTE);
Serial1.print(color & 0xFF, BYTE);

    OLED_GetResponse(PORT_1);

}
else if (puerto==PORT_2){
Serial2.print(OLED_TEXTFORMATED, BYTE);

        Serial2.print(MyChar, BYTE);
    Serial2.print(column, BYTE);
    Serial2.print(row, BYTE);


        // Color
Serial2.print(color >> 8, BYTE);
Serial2.print(color & 0xFF, BYTE);

    OLED_GetResponse(PORT_2);
}
}
```

A.1.2.Oled160.h (original)

```
/*************************************************

 uOLED-160-GMD1 Driver for Arduino
 Code: Oscar Gonzalez
 August 2007

 www.bricogeek.com

Modified by:Burcum Turkmen
May 2010

www.burcumturkmen.com
 *************************************************/

#include <stdint.h>
#include <WProgram.h>
#include <stdio.h>

#define OLED_BAUDRATE                   9600//57600
#define OLED_RESETPIN                   8     // PIN of reset
#define OLED_INITDELAYMS                5000

// Initialisation routine
#define OLED_DETECT_BAUDRATE                    0x55

// Drawing routines
#define OLED_CLEAR                      0x45
#define OLED_BKGCOLOR                   0x42
#define OLED_COPYPASTE                  0x63

#define OLED_LINE                       0x4C
#define OLED_CIRCLE                     0x43
#define OLED_CIRCLEFILL                 0x69
#define OLED_PUTPIXEL                   0x50
#define OLED_READPIXEL                  0x52
#define OLED_RECTANGLE                  0x72
#define OLED_PAINTAREA                  0x70

// Text properties
#define OLED_SETFONTSIZE                0x46
 #define        OLED_FONT5X7            0x01
 #define        OLED_FONT8X8            0x02
 #define        OLED_FONT8X12           0x03
#define OLED_TEXTFORMATED               0x54

// OLED Control
#define OLED_COMMAND_CONTROL                    0x59
```

```
#define OLED_COMMAND_DISPLAY                    0x01
#define OLED_COMMAND_CONTRAST                   0x02
#define OLED_COMMAND_POWER                      0x03

#define OLED_ACK  0x06  // Ok
#define OLED_NAK  0x15  // Error

void OLED_ResetDisplay()
{
  digitalWrite(OLED_RESETPIN, 0);
  delay(20);
  digitalWrite(OLED_RESETPIN, 1);
  delay(20);

}

char OLED_GetResponse()
{
  byte incomingByte = OLED_ACK;

  // Wait for data avaliable
  while (!Serial.available()) { delay(1); }

  // Read incoming byte
  incomingByte = Serial.read();

  return incomingByte;
}

// Initialise OLED display. You must first activate a serial comunication!
void OLED_Init()
{
  // First reset display
  OLED_ResetDisplay();

  delay(OLED_INITDELAYMS);              // Wait for init
  // Autodetect baudrate
  Serial.print(OLED_DETECT_BAUDRATE, BYTE);
  OLED_GetResponse();

}

// Get 16bits value from RGB (0 to 63, 565 format)
int GetRGB(int red, int green, int blue)
{

        char text[255];

        int outR = ((red * 31) / 255);
        int outG = ((green * 63) / 255);
```

```
        int outB = ((blue * 31) / 255);

        return (outR << 11) | (outG << 5) | outB;
}

void OLED_Clear()
{
 Serial.print(OLED_CLEAR, BYTE);
        //printByte(OLED_CLEAR); // Pixel write
delay(20);
    OLED_GetResponse();

}

void OLED_PutPixel(char x, char y, int color)
{

        // Putpixel
Serial.print(OLED_PUTPIXEL, BYTE);
Serial.print(x, BYTE);
Serial.print(y, BYTE);

        // Color
Serial.print(color >> 8, BYTE);
Serial.print(color & 0xFF, BYTE);//LBS


    OLED_GetResponse();

}

void OLED_DrawLine(char x1, char y1, char x2, char y2, int color)
{

        // Line drawing
Serial.print(OLED_LINE, BYTE);
Serial.print(x1, BYTE);
Serial.print(y1, BYTE);
Serial.print(x2, BYTE);
Serial.print(y2, BYTE);

        // Color
Serial.print(color >> 8, BYTE);
Serial.print(color & 0xFF, BYTE);// MSB


    OLED_GetResponse();

}
```

```
void OLED_DrawRectangle(char x, char y, char width, char height, char filled, int color)
{

        //int color = 0xFFFF;

        // Rectangle drawing
Serial.print(OLED_RECTANGLE, BYTE);
Serial.print(x, BYTE);
Serial.print(y, BYTE);
Serial.print(x+width, BYTE);
Serial.print(y+height, BYTE);

        // Color
Serial.print(color >> 8, BYTE);
Serial.print(color & 0xFF, BYTE);

        /*
        if (filled == 1) { printByte(0x01); }          // Filled
        else { printByte(0x00); }                              // Outline
        */
        OLED_GetResponse();
}

void OLED_DrawCircle(char x, char y, char radius, char filled, int color)
{    Serial.print(OLED_CIRCLE, BYTE);

        Serial.print(x, BYTE);
Serial.print(y, BYTE);
Serial.print(radius, BYTE);

        // Color
Serial.print(color >> 8, BYTE);
Serial.print(color & 0xFF, BYTE);


     OLED_GetResponse();

}

/*
 Change font format:
 FontType can be:
   OLED_FONT5X7
   OLED_FONT8X8
   OLED_FONT8X12
*/
void OLED_SetFontSize(char FontType)
{Serial.print(OLED_SETFONTSIZE, BYTE);
   Serial.print(FontType, BYTE);
```

```
   OLED_GetResponse();
}

void OLED_DrawText(char column, char row, char font_size, char *mytext, int color)
{
        //char mytext[] = "Arkanoid by pK";
Serial.print(0x73, BYTE);


    // Adjust to center of the screen (26 Columns at font size 0)
    int newCol = 13 - (strlen(mytext)/2);

    Serial.print(column, BYTE);
    Serial.print(row, BYTE);
Serial.print(font_size, BYTE);
        //printByte(font_size); // font size (0 = 5x7 font, 1 = 8x8 font, 2 = 8x12 font)

        // Color
Serial.print(color >> 8, BYTE);
Serial.print(color & 0xFF, BYTE);


        for (int i=0 ; i<strlen(mytext) ; i++)
        {
 Serial.print(mytext[i], BYTE);

        }
Serial.print(0x00, BYTE);


    OLED_GetResponse();

}

void OLED_DrawSingleChar(char column, char row, char font_size, char MyChar, int color)
{
    Serial.print(OLED_TEXTFORMATED, BYTE);

        Serial.print(MyChar, BYTE);
    Serial.print(column, BYTE);
    Serial.print(row, BYTE);


        // Color
Serial.print(color >> 8, BYTE);
Serial.print(color & 0xFF, BYTE);

    OLED_GetResponse();

}
```

A.4.LIBRERÍA VALORES.H

Librería donde se encuentran definidas las constantes y valores que se usan en el programa principal.

```
/* LIBRERÍA LORENA */
//gifs
#define _parpadeo              0
#define _riendose              1
#define _noriendose            2
#define _sorprendido           3
#define _nosorprendido         4
#define _triste                5
#define _notriste              6
#define _sospechoso            7
#define _normal                8
#define _enfadandose           9
#define _noenfadandose         10
#define _abriendob             11
#define _cerrandob             12
#define _abriendo              13
#define _cerrando              14


//tiempos gifs


#define TiempoParpadeo                 1520
#define TiempoRiendose                 1500
#define TiempoNoriendose               1500
#define TiempoSorprendido              400
#define TiempoNosorprendido            400
#define TiempoTriste                   2000
#define TiempoNotriste                 1500
#define TiempoSospechoso               2500
#define TiempoNormal                   400
#define TiempoEnfadandose              2000
#define TiempoNoenfadandose            2000
#define TiempoAbriendob                1275
#define TiempoCerrandob                1275
#define TiempoAbriendo                 1200
```

```
#define TiempoCerrando                  1200

char parpadeo_entero      []      =
   {0x40,0x56,0x00,0x80,0x80,0x10,0x50,0x00,0x13,0x00,0x0E,0x40};
char riendose_entero      []      =
   {0x40,0x56,0x00,0x80,0x80,0x10,0x96,0x00,0x0A,0x00,0x1E,0x80};
char noriendose_entero    []      =
   {0x40,0x56,0x00,0x80,0x80,0x10,0x96,0x00,0x0A,0x00,0x1C,0x00};
char sorprendido_entero   []      =
   {0x40,0x56,0x00,0x80,0x80,0x10,0x32,0x00,0x08,0x00,0x1A,0x00};
char nosorprendido_entero[]       =
   {0x40,0x56,0x00,0x80,0x80,0x10,0x32,0x00,0x08,0x00,0x18,0x00};
char triste_entero []     =
   {0x40,0x56,0x00,0x80,0x80,0x10,0xC8,0x00,0x0A,0x00,0x15,0x80};
char notriste_entero      []      =
   {0x40,0x56,0x00,0x80,0x80,0x10,0x96,0x00,0x0A,0x00,0x0A,0x80};
char sospechoso_entero    []      =
   {0x40,0x56,0x00,0x80,0x80,0x10,0xFA,0x00,0x0A,0x00,0x13,0x00};
char normal_entero        []      =
   {0x40,0x56,0x00,0x80,0x80,0x10,0x50,0x00,0x05,0x00,0x0D,0x00};
char enfadandose_entero   []      =
   {0x40,0x56,0x00,0x80,0x80,0x10,0xC8,0x00,0x0A,0x00,0x08,0x00};
char noenfadandose_entero[]       =
   {0x40,0x56,0x00,0x80,0x80,0x10,0xC8,0x00,0x0A,0x00,0x05,0x80};
char abriendob_entero     []      =
   {0x40,0x56,0x00,0x80,0x80,0x10,0xFF,0x00,0x05,0x00,0x01,0x80};
char cerrandob_entero     []      =
   {0x40,0x56,0x00,0x80,0x80,0x10,0xFF,0x00,0x05,0x00,0x04,0x40};
char abriendo_entero      []      =
   {0x40,0x56,0x00,0x80,0x80,0x10,0xC8,0x00,0x06,0x00,0x00,0x00};
char cerrando_entero      []      =
   {0x40,0x56,0x00,0x80,0x80,0x10,0xC8,0x00,0x06,0x00,0x02,0xC0};

int parpadeo        []      =       {0x50,0x0013,0x000E40};
int riendose        []      =       {0x96,0x000A,0x001E80};
int noriendose      []      =       {0x96,0x000A,0x001C00};
int sorprendido     []      =       {0x32,0x0008,0x001A00};
int nosorprendido[]         =       {0x32,0x0008,0x001800};
int triste          []      =       {0xC8,0x000A,0x001580};
int notriste        []      =       {0x96,0x000A,0x000A80};
int sospechoso      []      =       {0xFA,0x000A,0x001300};
int normal          []      =       {0x50,0x0005,0x000D00};
int enfadándose     []      =       {0xC8,0x000A,0x000800};
int noenfadandose[]         =       {0xC8,0x000A,0x000580};
```

```
int abriendob     []     =     {0xFF,0x0005,0x000180};
int cerrandob     []     =     {0xFF,0x0005,0x000440};
int abriendo      []     =     {0xC8,0x0006,0x000000};
int cerrando      []     =     {0xC8,0x0006,0x0002C0};

char *mensaje_parpadeo[11]={"parpadeando"};
char *mensaje_riendose[7]={"riendo"};
char *mensaje_noriendose[10]={"no riendo"};
char *mensaje_sorprendido[16]={"sorprendiendose"};
char *mensaje_nosorprendido[19]={"no sorprendiendose"};
char *mensaje_triste[17]={"entristeciendose"};
char *mensaje_notriste[20]={"desentristeciendose"};
char *mensaje_sospechoso[12]={"sospechando"};
char *mensaje_normal[7]={"normal"};
char *mensaje_enfadado[12]={"enfadandose"};
char *mensaje_noenfadado[15]={"desenfadansose"};
char *mensaje_abriendo[9]={"abriendo"};
char *mensaje_cerrando[9]={"cerrando"};
```

A.5. APLICACIÓN CLIENTE

```
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>

#define BAUDRATE B9600
#define MODEMDEVICEA "/dev/ardulore"

////////////////////////////////////////////////////////////////////////////////

int main(void)
{
  int fda;
  struct termios oldtioa, newtioa;
  char inbuff[500], outbuff[500];
  int lbuff, lrxbuff, lreciv, dato;
  int line = 0;
  char opcion;

////////////////////////////////// Abrimos puerto

  fda = open(MODEMDEVICEA, O_RDWR | O_NOCTTY);
  if(fda < 0){
    perror(MODEMDEVICEA);
    exit(-1);
  }
  else printf("\nPuerto 1 abierto. \n");


//////////////////////////////////Captamos configuración puerto serie y la guardamos

  tcgetattr(fda, &oldtioa);

//////////////////////////////////Configuramos puerto serie

  bzero(&newtioa, sizeof(newtioa));
  newtioa.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
  newtioa.c_iflag = IGNPAR;
  newtioa.c_oflag = 0;
  newtioa.c_lflag = 0;

  newtioa.c_cc[VTIME] = 1;

  newtioa.c_cc[VMIN]  = 0;

//////////////////////////////////Mandamos configuración al puerto serie

 tcsetattr(fda, TCSANOW, &newtioa);
```

```
printf("\nIntroduzca numero del comando. \n");
printf("\nPulse 1 para parpadear \n");
printf("\nPulse 2 para reir \n");
printf("\nPulse 3 para sospechar por pantalla izquierda \n");
printf("\nPulse 4 para sorprenderse por pantalla derecha \n");
printf("\nPulse 5 para permanecer triste \n");
printf("\nPulse 6 para sospechar \n");
printf("\nPulse 7 para sorprenderse \n");
printf("\nPulse 8 para permanecer sonriendo \n");
printf("\nPulse 9 para dejar de sonreír \n");


while(1){
	scanf("%d",&dato);
	if(dato == 0) break;
	printf("El numero introducido es: %d\n",dato);

	switch (dato)
	{

		case 1: //parpadeo constante

		//Construimos comando

		outbuff[0]='D'; //Función
		outbuff[1]='P'; //Función
		outbuff[2]='0'; //Pantalla
		outbuff[3]='0'; //Animación a
		outbuff[4]='0'; //Animación b
		outbuff[5]='2'; //tiempo extra
		outbuff[6]='3'; //Estado
		outbuff[7]='\r';//Retorno de carro

		lbuff = strlen(outbuff) + 1;

		//Enviamos comando
		write(fda, outbuff, lbuff);

		printf("\nParpadeando \n");

		break;

		case 2: //reir y opuesto

		//Construimos comando

		outbuff[0]='D'; //Función
		outbuff[1]='P'; //Función
		outbuff[2]='0'; //Pantalla
		outbuff[3]='0'; //Animación a
```

```
outbuff[4]='1'; //Animación b
outbuff[5]='2'; //tiempo extra
outbuff[6]='1'; //Estado
outbuff[7]='\r';//Retorno de carro

lbuff = strlen(outbuff) + 1;

//Enviamos comando
write(fda, outbuff, lbuff);
printf("\nRiendo \n");
break;


case 3: //sospechar por la pantalla izquierda

//Construimos comando

outbuff[0]='D'; //Función
outbuff[1]='P'; //Función
outbuff[2]='1'; //Pantalla
outbuff[3]='0'; //Animación a
outbuff[4]='7'; //Animación b
outbuff[5]='0'; //tiempo extra
outbuff[6]='3'; //Estado
outbuff[7]='\r';//Retorno de carro


lbuff = strlen(outbuff) + 1;

//Enviamos comando
write(fda, outbuff, lbuff);
printf("\nSospechando \n");
break;

case 4: //Sorprenderse por la pantalla derecha

//Construimos comando

outbuff[0]='D'; //Función
outbuff[1]='P'; //Función
outbuff[2]='2'; //Pantalla
outbuff[3]='0'; //Animación a
outbuff[4]='3'; //Animación b
outbuff[5]='2'; //tiempo extra
outbuff[6]='1'; //Estado
outbuff[7]='\r';//Retorno de carro

lbuff = strlen(outbuff) + 1;

//Enviamos comando
```

```
write(fda, outbuff, lbuff);
printf("\nSorprendiéndose \n");
break;

case 5: //permanecer triste

//Construimos comando

outbuff[0]='D'; //Función
outbuff[1]='P'; //Función
outbuff[2]='0'; //Pantalla
outbuff[3]='0'; //Animación a
outbuff[4]='5'; //Animación b
outbuff[5]='0'; //tiempo extra
outbuff[6]='4'; //Estado
outbuff[7]='\r';//Retorno de carro

lbuff = strlen(outbuff) + 1;

//Enviamos comando
write(fda, outbuff, lbuff);
printf("\nEntristeciéndose \n");
break;

case 6: //sospecha por las dos pantallas

//Construimos comando

outbuff[0]='D'; //Función
outbuff[1]='P'; //Función
outbuff[2]='0'; //Pantalla
outbuff[3]='0'; //Animación a
outbuff[4]='7'; //Animación b
outbuff[5]='0'; //tiempo extra
outbuff[6]='2'; //Estado
outbuff[7]='\r';//Retorno de carro

lbuff = strlen(outbuff) + 1;

//Enviamos comando
write(fda, outbuff, lbuff);
printf("\nSospechando \n");
break;

case 7: //Sorprenderse en las dos pantallas

//Construimos comando
outbuff[0]='D'; //Función
outbuff[1]='P'; //Función
outbuff[2]='0'; //Pantalla
```

```
outbuff[3]='0'; //Animación a
outbuff[4]='3'; //Animación b
outbuff[5]='2'; //tiempo extra
outbuff[6]='1'; //Estado
outbuff[7]='\r';//Retorno de carro

lbuff = strlen(outbuff) + 1;

//Enviamos comando
write(fda, outbuff, lbuff);
printf("\nSorprendiéndose \n");
break;

case 8: //Permanecer sonriendo en las dos pantallas

//Construimos comando

outbuff[0]='D'; //Función
outbuff[1]='P'; //Función
outbuff[2]='0'; //Pantalla
outbuff[3]='0'; //Animación a
outbuff[4]='1'; //Animación b
outbuff[5]='0'; //tiempo extra
outbuff[6]='4'; //Estado
outbuff[7]='\r';//Retorno de carro

lbuff = strlen(outbuff) + 1;

//Enviamos comando

write(fda, outbuff, lbuff);
printf("\nSonriendo \n");
break;

case 9: //Dejar de sonreír

//Construimos comando

outbuff[0]='p'; //Función
outbuff[1]='P'; //Función
outbuff[2]='0'; //Pantalla
outbuff[3]='0'; //Animación a
outbuff[4]='2'; //Animación b
outbuff[5]='0'; //tiempo extra
outbuff[6]='5'; //Estado
outbuff[7]='\r';//Retorno de carro

lbuff = strlen(outbuff) + 1;

//Enviamos comando
```

```
                write(fda, outbuff, lbuff);
                printf("\nDejando de sonreír \n");
                break;

        } //salida del switch

    sleep(3);
        lreciv = read(fda, inbuff, lbuff);


   if(lreciv>0)
     printf("dato leido: %s\n",inbuff);



} //salida del while

////////////////////////////////////Restauramos configuración original del puerto

  tcsetattr(fda, TCSANOW, &oldtioa);

////////////////////////////////////Cerramos puerto

  close(fda);
}
```