Trabajo Fin de Máster
Máster Universitario en Ingenería Electrónica, Robótica y Automática

Sistema de grabación de señales PPM para aplicaciones de animatrónica

Autor: Carlos García de la Torre

Tutor: Jose María Maestre Torreblanca

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Máster
Máster Universitario en Ingeniería Electrónica, Robótica y Automática

Sistema de grabación de señales PPM para aplicaciones de animatrónica

Autor:
Carlos García de la Torre

Tutor:
José María Maestre Torreblanca
Profesor titular

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo Fin de Máster: Sistema de grabación de señales PPM para aplicaciones de animatrónica

Autor: Carlos García de la Torre

Tutor: José María Maestre Torreblanca

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A mis abuelos Antonio, Cristina, Carlos y Anita.

Agradecimientos

Me gustaría agradecer a mi tutor, Jose María Maestre Torreblanca, su labor, su dedicación y su apoyo, moral y académico durante todo el desarrollo del proyecto.

Querría también agradecer a mis padres, Ignacio y Ana, y a mi hermano, Alberto, su paciencia y su ayuda que me han servido para mantener el ánimo en los momentos difíciles y han celebrado conmigo los avances en este proceso de aprendizaje y estudio.

Agradecer también al resto del personal docente los meses dedicados a mi formación y desarrollo personal.

Carlos García de la Torre

Sevilla, 2017

Resumen

El objetivo de este trabajo es tratar de reflejar el estudio y el proceso de desarrollo de un sistema de grabación de señales PPM enfocado a aplicaciones de control de prototipos de robótica y animatrónica.

Se comienza realizando un estudio de las señales PPM y PWM que se van a tratar a nivel práctico para describir la línea de trabajo a seguir. También se realiza una breve descripción de las diferentes plataformas de hardware contempladas, destacando los microcontroladores Arduino como opción de bajo coste.

Se contemplan varias opciones de montaje y se analizan los resultados para obtener una solución con una buena reproducción de la señal grabada.

Tras realizar pruebas funcionales, caracterizar datos y trabajar sobre la solución final, se obtienen resultados y patrones que demuestran un correcto tratamiento de los datos obtenidos y una grabación fiel al control que se desea realizar.

Por último se plantean líneas futuras de desarrollo y mejora de cara a una aplicación práctica de este tipo de control grabado.

Abstract

The aim of this work is to try to show the research and development process of a PPM signal recording system focused on control applications for robotics and animatronics prototype.

It begins with a review of the PPM and PWM signals that will be to describe the work in process path. There is also a brief description of different hardware platforms considered, highlighting Arduino microcontrollers as a low cost option.

Several mounting options are considered and results are analyzed in order to obtain a good solution and a recorded signal acceptable.

After performing functional tests, getting data and working on the final solution, we obtain results and patterns that demonstrate a correct data treatment and a faithful recording.

Finally, future lines of development and improvement are proposed for a practical application of this type of recorded control.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

NOTACIÓN

s	Segundos
ms	Milisegundos
µs	Microsegundos
Kb	Kilobytes
Mb	Megabytes
Gb	Gigabytes
tr	Trama
R/C	Radiocontrol

1 INTRODUCCIÓN

1.1 Motivación del Proyecto

Este proyecto pretende culminar una etapa muy importante en mi carrera. Tras realizar mis estudios de ingeniería comprobé que el mundo laboral requiere ciertos conocimientos adquiridos con la experiencia, pero, otros conocimientos, requieren una formación muy específica, como es el caso de la robótica y la automatización. Me gustaba, era lo que quería hacer y, de esta manera, tras conocer la industria a través de diferentes trabajos, he aprovechado la oportunidad que me brinda este trabajo fin de máster para desarrollar un sistema de control que me permita conocer a fondo cómo funciona lo que antes no entendía demasiado bien: la automatización y el control de robots.

No tenía muy claro como enfocar el trabajo, pero todo quedó mas claro después de una charla, en una de las clases, acerca del empleo de la robótica en el cine, en la que durante unas pocas horas fui entendiendo que quería realizar alguna aportación, por mínima que fuera, a este mundo.

Así, tras hablar con mi tutor, decidimos cubrir una necesidad que anteriormente nos había trasladado el ponente de la charla: Un grabador – reproductor de señales de control. Un sistema que, recibiendo señales de control enviadas por un mando, que a su vez es manejado por una persona, sea capaz de almacenar estas señales de control y volver a repetir los mismos movimientos y patrones anteriormente grabados. De la misma forma que se graban las escenas de las películas.

El objetivo es que, el robot, entre en la dinámica de rodaje y pueda afrontar varias tomas, correcciones, cambios de guión y, a diferencia de un actor, ser capaz de repetir exactamente la toma válida con independencia del resto del rodaje. También se busca la posibilidad de poder grabar alguna toma previa de la escena para que los actores puedan, después, rodar sabiendo exactamente qué hará el robot.

En este proyecto se ha querido hacer una primera aproximación investigando la grabación y generación de las señales PPM de control con las que trabajan este tipo de robots.

1.2 Efectos especiales y animatrónica

En la industria del cine, el teatro y el entretenimiento en general, muy a menudo, se trata de simular situaciones o escenarios que difícilmente se dan en la vida real con ciertas condiciones de seguridad para los actores. Este es el principal reto que se propone la industria de los efectos especiales.

Se han hecho grandes avances técnicos tratando de recrear escenarios, localizaciones y situaciones imposibles en los que se llegan a realizar grandes esfuerzos para simular las condiciones que se darían en estos escenarios y para que los actores puedan trabajar sin correr peligro alguno. Explosiones controladas, conducción extrema, caídas, vuelos, seres que no existen o no podrían actuar como se desea, y muchos otros fenómenos que con ciertas técnicas se pueden llegar a trasladar al espectador como un fenómeno real.

En este sentido, la animatrónica consiste en la creación de estructuras, máquinas y mecanismos que simulan, con mayor o menor complejidad, seres animados con vida propia. Tiene su origen en la creación de marionetas y títeres, e involucra a ingenieros, artesanos, artistas, etc.

En el caso de la animatrónica, se aplican conceptos mecatrónicos (mecánicos – electrónicos) para crear dispositivos o seres que actúen como animales, personas, plantas u objetos que cobran vida.

Se utilizan en muchas aplicaciones, entre ellas, parques de atracciones, teatro y su aplicación más conocida en la actualidad es el cine. Aunque ya se realizan muchas escenas con ayuda de efectos digitales generados por ordenador, se sigue trabajando ampliamente con multitud de animatronics que están en contacto directo con actores y que se convierten, en muchos casos, en parte del reparto.

Debido a la complejidad de muchos de los animatronics empleados en el rodaje de películas, la animatrónica se considera la rama de la robótica que introduce robots, cada vez más sofisticados, en el mundo de la actuación y los efectos especiales.

1.3 Aplicación

Este trabajo se centra en tratar de abordar una cuestión importante en el desarrollo de la animatrónica, cuyo resultado puede ser extrapolable a otras técnicas de control.

Tradicionalmente los robots animatrónicos son controlados por los llamados “titiriteros”. Personas cuya misión es manejar el robot de tal manera que simule, de la mejor forma posible, un comportamiento natural del ser animado. En otras palabras, el titiritero controla el robot de tal forma que no parezca un robot y actúe como el ser que se pretende simular.

Este tipo de control es muy difícil de programar, ya que en muchas ocasiones, el animatronic actúa directamente con personas, y tiene que reaccionar ante estímulos y acciones de los actores que se producen durante la escena, por lo que se sigue realizando de forma manual durante la escena.

El objetivo de este trabajo es el de grabar y reproducir la señal de control enviada por el titiritero para poder repetir los mismos movimientos como si de una rutina de control se tratase, aprovechando las ventajas del control manual y la comodidad de repetir, editar y corregir este control las veces que sea necesario.

1.4 Plataforma Arduino. Hardware Libre

Para este proyecto se ha optado por utilizar como plataforma física los dispositivos y microcontroladores desarrollados por Arduino.

La empresa Arduino ha sido la encargada de llevar a la práctica un concepto bastante novedoso en lo que a plataformas hardware se refiere. Hasta ahora solo en el desarrollo y consumo de software se podía aplicar el concepto de “software libre”, disponiendo así de un entorno colaborativo de desarrollo que ha acelerado muchas investigaciones y el desarrollo de la tecnología.

Arduino ha aplicado este mismo concepto de “libre” a las plataformas hardware cediendo sus diseños, su know-how y vendiendo sus plataformas a un coste reducido para permitir este mismo entorno colaborativo que se daba en el software libre a la comunidad de desarrolladores de hardware. También ha creado un lenguaje de programación de alto nivel con librerías que permiten una programación sencilla.

Su producto más vendido es el Arduino UNO, que ha sido utilizado en este trabajo entre otros dispositivos. Debido al gran impacto de esta plataforma, la información necesaria para desarrollar prototipos en este soporte

está documentada ampliamente en la red y es de dominio público. Además de multitud de comunidades y empresas que se dedican a asesorar y desarrollar proyectos con Arduino, la propia empresa ofrece varias webs de asistencia técnica, tutoriales y consultas entre los propios usuarios.

Estas razones han sido suficientes para trabajar con estos dispositivos en este proyecto. Además de la posibilidad que brinda la plataforma de elaborar una solución de bajo coste.

2 PWM, PPM Y SERVOS

Uno de los principales actuadores en prototipos de robótica son los servomotores, conocidos también como servos. Aunque se utilizan muchos otros actuadores en robots animatrónicos muchos de ellos, al igual que los servomotores, son controlados con señales de Modulación por Ancho de Pulso o PWM.

De cara a la comunicación entre el control y los diferentes dispositivos actuadores que intervienen en el sistema se utiliza una señal de control única que agrupa varias señales PWM por canales. Esta señal es de Modulación por Posición de Pulso, o señal PPM.

2.1. Servomotores

Conocidos también como servos, los servomotores traducen la señal de entrada a una posición angular concreta de su eje principal, de tal manera que al variar la señal se puede controlar el ángulo y la velocidad de giro del eje.

Este tipo de motores se utiliza principalmente en las articulaciones y ejes de movimiento de todo tipo de dispositivos y, en nuestro caso particular, son los que mueven las articulaciones de los robots animatrónicos, por lo que otro parámetro importante a tener en cuenta es el par mecánico que este motor es capaz de aplicar a un mecanismo.

Aunque es verdad que los robots tienen todo tipo de actuadores neumáticos, hidráulicos, dispositivos sonoros, luminosos, etcétera, los servomotores ocupan gran parte del prototipado y desarrollo de mecanismos que no requieren un excesivo esfuerzo mecánico.

La mayoría de los servomotores funcionan con corriente continua, aunque también existen los que funcionan con corriente alterna.

En el caso de este trabajo, los servomotores utilizados en las pruebas son un elemento meramente ilustrativo que no requerirá ningún valor de par o velocidad específicos. Simplemente se utilizarán para ilustrar de forma más visual el trabajo que se está llevando a cabo con la señal de control.

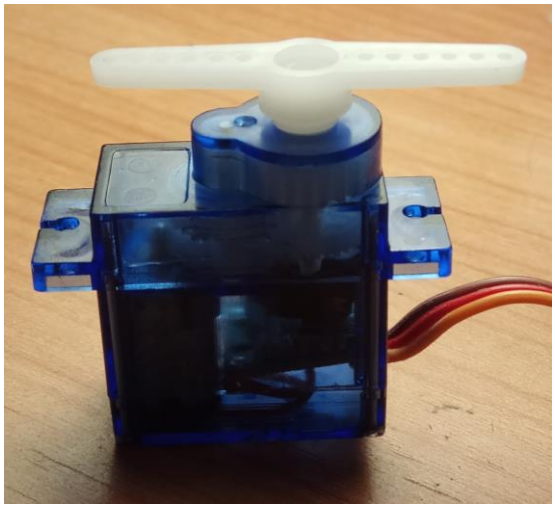


Figura 2-1. Servomotor.

2.1.1 Parámetros principales

A la hora de seleccionar un servomotor de cara a una aplicación concreta conviene saber previamente el rango en el que se moverán los siguientes parámetros:

- Par mecánico:

  Expresado en Kg/cm, es la fuerza máxima ejercida por el servomotor. Esta unidad hace referencia a los kilos que puede mover el sevomotor con un eje de 1 cm de radio.

  En los servos, el rango de par es bastante amplio dependiendo del tamaño y los mecanismos, no obstante, el par es directamente proporcional a la corriente consumida con lo que un servo más grande consumirá más energía para realizar un movimiento si la carga que soporta es mayor.

- Velocidad angular:

  Velocidad de rotación que el servomotor puede transmitir en el eje principal, medida en radianes por segundo (rad/s).

2.1.2 Partes constructivas del servomotor

Un servomotor no deja de ser un motor eléctrico controlado por un circuito que compara la posición del eje con la posición deseada mediante un potenciómetro. Este circuito aplica una corriente al motor proporcional al par que se ejerce para que el eje alcance la posición deseada. Para este proceso son necesarios los siguientes elementos:

- Motor eléctrico: encargado de generar par sobre el eje.
- Sistema de regulación: Engranajes que actúan sobre el motor para regular la velocidad y el par. Mediante estos engranajes se puede actuar sobre estos parámetros con exactitud.
- Un sistema de control o sensor: circuito electrónico que controla el movimiento del motor mediante el envío de pulsos eléctricos.
- Un potenciómetro: conectado al eje central del motor que indica el ángulo en el que se encuentra el eje del motor.

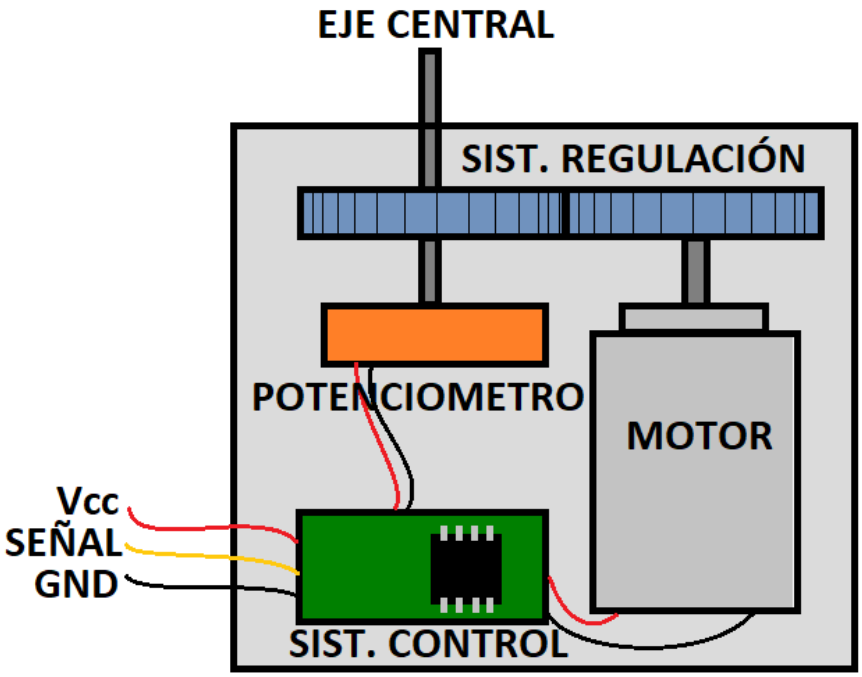


Figura 2-2. Partes de un servomotor.

2.2. Señal Pulse Width Modulation – PWM

La señal PWM o Señal Modulada por Ancho de Pulso es la señal con la que se controlan muchos de los actuadores que intervienen en robótica y, también, en muchos otros dispositivos que son utilizados de forma cotidiana en instalaciones y en la industria. Se usa, por ejemplo, para modificar la emisión de un diodo LED variando la intensidad de luz de forma casi instantánea.

En el caso concreto de la aplicación que se trata en esta memoria, los servomotores se controlan con este tipo de señal como se expondrá a continuación.

Figura 2-3. Captura de analizador lógico: Señal Modulada por Ancho de Pulso – PWM.

La señal PWM es una señal periódica cuyo ciclo de trabajo es modificado para transmitir una información o controlar la energía enviada a una carga. Esta señal, como su propio nombre indica, se centra en la duración (o ancho) de los pulsos, de tal manera que a mayor duración del pulso, mayor valor tendrá la variable identificada.

En otras palabras, se establece un periodo estándar en el cual la señal podrá estar a nivel alto o bajo. Dependiendo del porcentaje del periodo en el que la señal este a un nivel o a otro se obtendrá un valor proporcional al porcentaje que ocupa este pulso dentro del período de la señal.

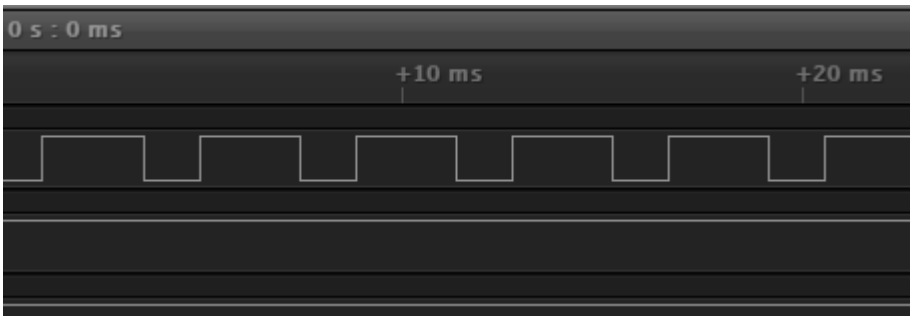


Figura 2-4. Captura de analizador lógico: Ejemplo de ciclo de trabajo del 100 %. Señal PWM.

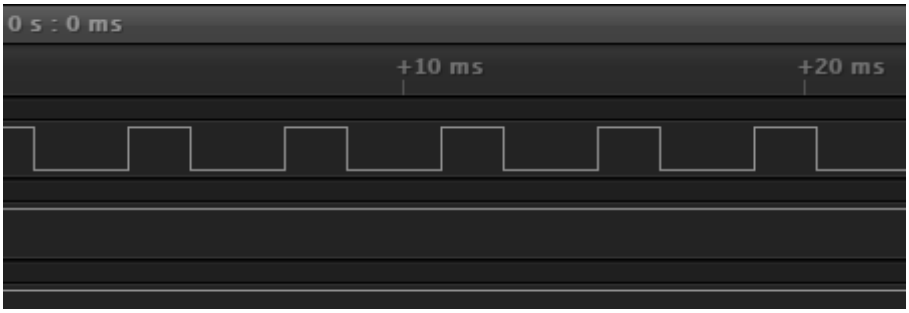


Figura 2-5. Captura de analizador lógico: Ejemplo de ciclo de trabajo del 50 %. Señal PWM.

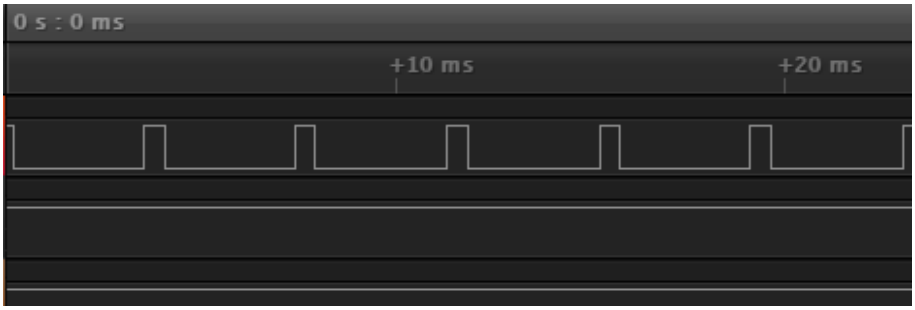


Figura 2-6. Captura de analizador lógico: Ejemplo de ciclo de trabajo del 0 %. Señal PWM.

El periodo de la señal PWM debe ser siempre el mismo independientemente del ciclo de trabajo de los pulsos. Solo así se puede medir el porcentaje del tiempo en el que la señal está activa.

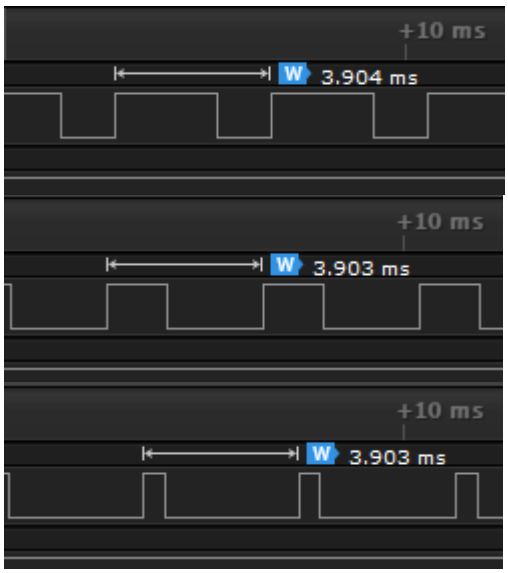


Figura 2-7. Captura de analizador lógico: Periodo de la señal independiente del ciclo de trabajo.

La señal PWM utilizada en esta aplicación suele ser cuadrada y suele estar por defecto a nivel bajo. El valor viene dado por el porcentaje del periodo en el que la señal se encuentra a nivel alto. Sin embargo, siempre tiene que existir un valor mínimo de la señal a nivel alto para que exista control, de lo contrario una señal continua a nivel bajo se identifica como que el dispositivo de control está apagado, o que no la señal de control no llega correctamente. En este caso, la señal PWM entregada a los servos variará entre 1 ms y 2 ms.

Un circuito PWM se realiza usando un comparador de dos entradas y una salida. Una de las entradas se conecta a un oscilador de onda dientes de sierra, mientras que la otra queda disponible para la señal moduladora. En la salida la frecuencia es generalmente igual a la de la señal dientes de sierra y el ciclo de trabajo está en función de la portadora.

Muchos de los actuadores utilizados en robótica se controlan en última instancia por señales PWM cuyos ciclos de trabajo son modificados para controlar la posición, la velocidad o la energía empleada en sus procesos de funcionamiento.

2.3. Pulse Position Modulation – PPM

La Señal Modulada por Posición de Pulso o señal PPM se compone de tramas de pulsos que se envían periódicamente, de tal manera que cada pulso contiene una información según su duración y su posición en la trama.

Figura 2-8. Captura de analizador lógico: Señales Moduladas por Posición de Pulso – PPM.

Esta señal se podría comprender mejor imaginando un conjunto de pulsos PWM agrupados en una trama y cuya posición dentro de la trama los identifica como un dato en concreto.

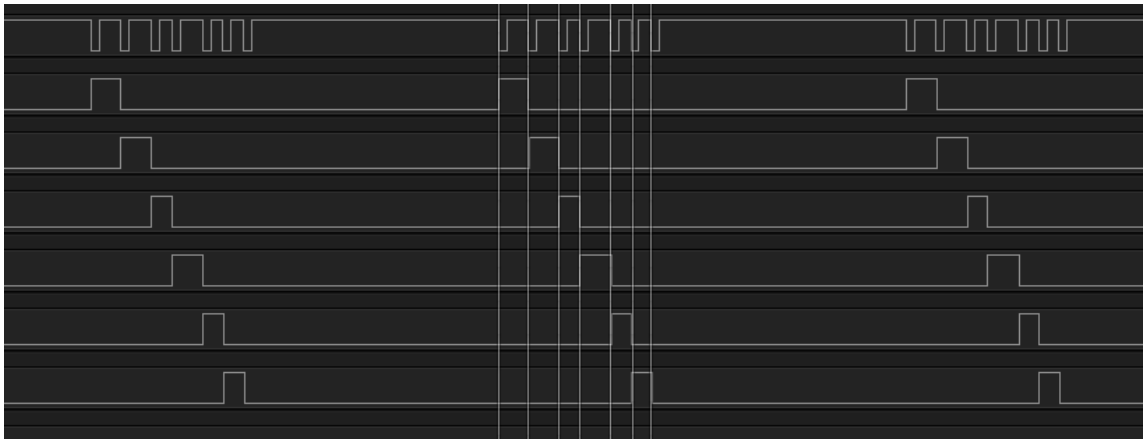


Figura 2-9. Captura de analizador lógico: Descomposición de señal PPM en señales PWM.

En la aplicación en la que se centra esta memoria, se trabaja con una señal PPM. Esta señal es generada por una emisora R/C.

Normalmente y de forma estándar, en esta tecnología se establecen varios parámetros que serán tenidos en cuenta más adelante.

2.3.1 Canal

Se identifica como canal la posición que ocupa el pulso dentro de la trama. El canal corresponde siempre a un mismo dato. El dispositivo emisor tendrá tantos canales como posiciones genere en cada trama.

El emisor y el receptor de esta señal deben trabajar con el mismo número de canales, y estos deben estar sincronizados, para que el dato generado en un canal, sea recibido en el mismo canal cuando llega la señal.

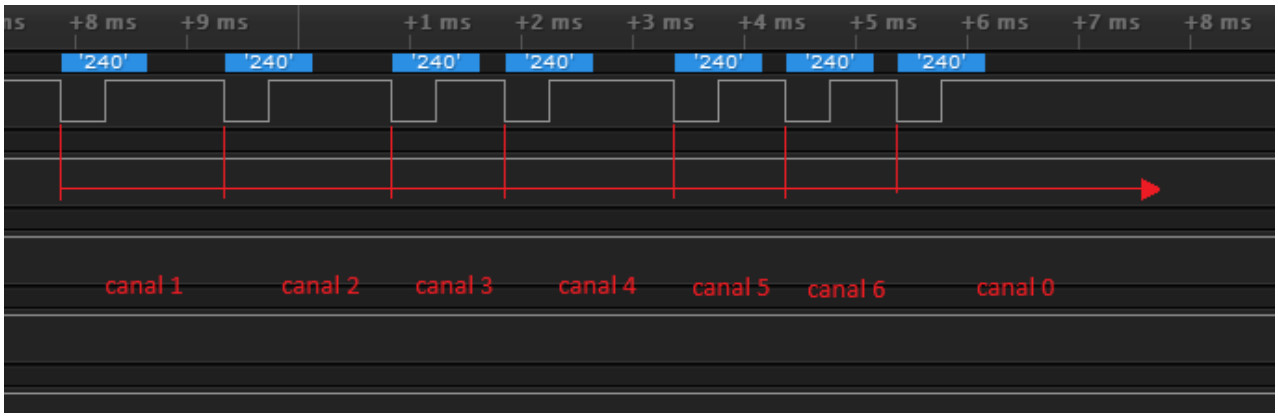


Figura 2-10. Captura de analizador lógico: División de señal PPM por canales.

2.3.2 Periodo

El periodo de la señal se establece a 20 ms, o a 30 ms si existe un número de canales que no puedan ser enviados en periodos de 20 ms. El periodo es estándar en la señal y no varía al igual que ocurría con el periodo de las señales PWM.

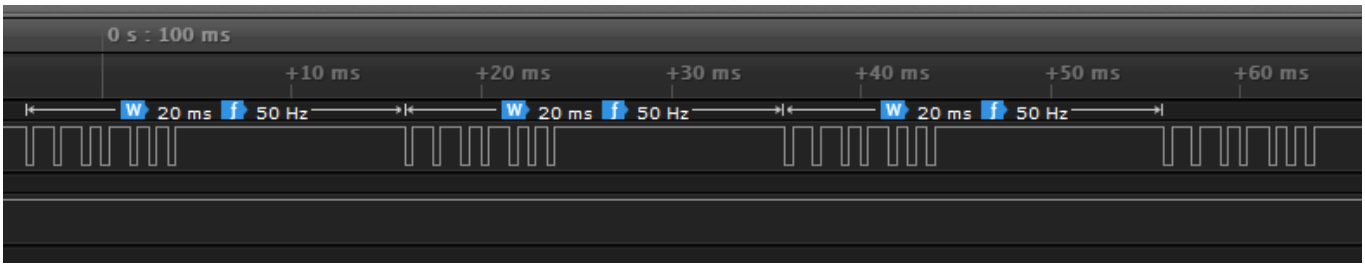


Figura 2-11. Captura de analizador lógico: Periodo de señal PPM.

2.3.3 Tiempo vacío de sincronización

Durante el periodo de la señal se debe contar con un tiempo suficiente para que se pueda diferenciar una trama de otra estableciendo un tiempo prudencial entre ellas para sincronizar los dispositivos. La duración de este tiempo será siempre la diferencia entre la duración de la trama y el periodo de la señal.

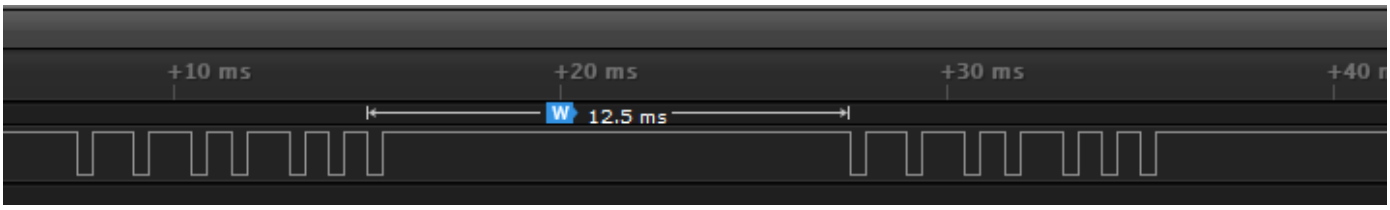


Figura 2-12. Captura de analizador lógico: Tiempo vacío de sincronización de señal PPM.

2.3.4 Duración del pulso

El intervalo de tiempo en el que se moverá la señal considerada para esta aplicación se encuentra entre 1 ms y 2 ms, siendo el ciclo de trabajo del 0 % para 1 ms y del 100 % para 2 ms.

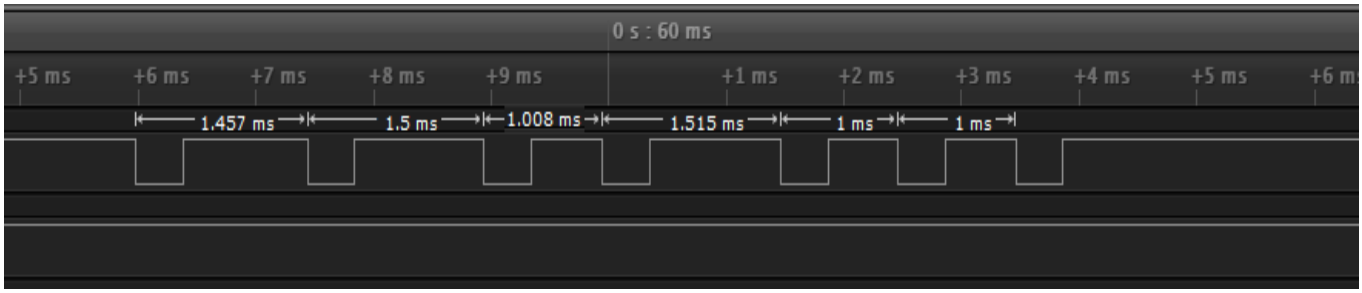


Figura 2-13. Captura de analizador lógico: Intervalo de tiempo efectivo de un pulso de una señal PPM.

2.3.5 Gaps de separación entre canales

Son pequeños espacios que separan un canal del siguiente. La duración de estos pequeños espacios es fija y se suma a la duración del pulso que viene a continuación.

Un gap y un pulso ocupan una posición o canal completo y la duración de este es la que finalmente se traduce en el valor del dato. En la aplicación tratada en esta memoria se trabaja con un gap de 0,4 ms.

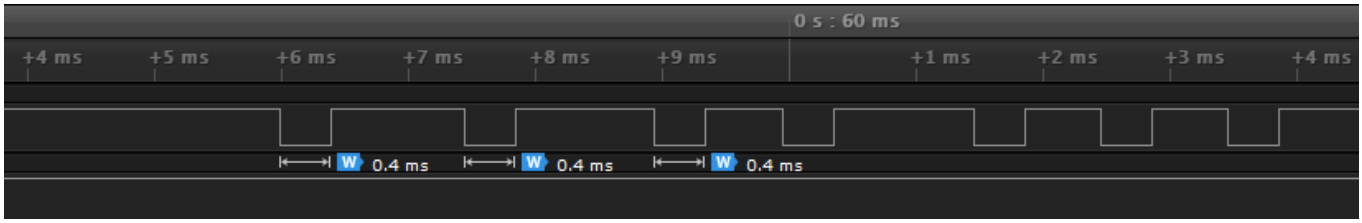


Figura 2-14. Captura de analizador lógico: Gaps de separación de una señal PPM.

2.3.6 Positivo o negativo

La señal PPM puede estar por defecto a nivel bajo o a nivel alto. En este sentido no existe un tipo predefinido. Debido a que los datos vienen determinados por intervalos temporales, siempre que la señal esté bien estructurada y sincronizada, se utilizan, indistintamente, dispositivos para señales PPM positivas o negativas, incluso algunos dispositivos admiten ambas posibilidades. No obstante, es necesario conocer previamente con qué tipo de señal PPM se está trabajando en este sentido.

Figura 2-15. Captura de analizador lógico: Señal PPM Positiva.

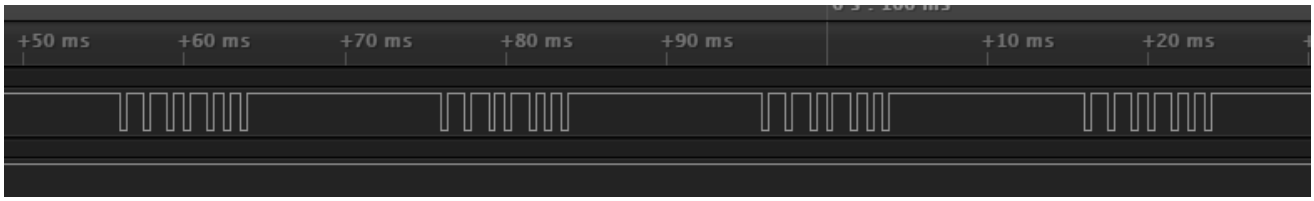


Figura 2-16. Captura de analizador lógico: Señal PPM Negativa o Inversa.

En concreto, la señal PPM inversa es la que emite la emisora con la que se realizará todo el control y los experimentos durante el desarrollo de las pruebas en este trabajo.

3 DESCRIPCIÓN DE HARDWARE

Este proyecto contempla una parte importante de implementación harware y software correspondiente a la investigación y el desarrollo de una solución viable para la grabación de señales de control en robots de animatrónica.

Atendiendo a las últimas tendencias en el campo del desarrollo de soluciones tecnológicas y a la necesidad de una implementación de bajo coste, se ha optado por la utilización de diferentes módulos de hardware basados en la plataforma Arduino.

También se emplea una emisora radiocontrol para la generación de señales de entrada al dispositivo, una tarjeta micro SD, varios servomotores pequeños con objeto de probar el prototipo y, cables, placas protoboard y accesorios para el conexionado durante las pruebas.

A continuación se enumeran los principales elementos de hardware utilizados en este proyecto, que pasarán a ser explicados a continuación.

3.1. Arduino YÚN

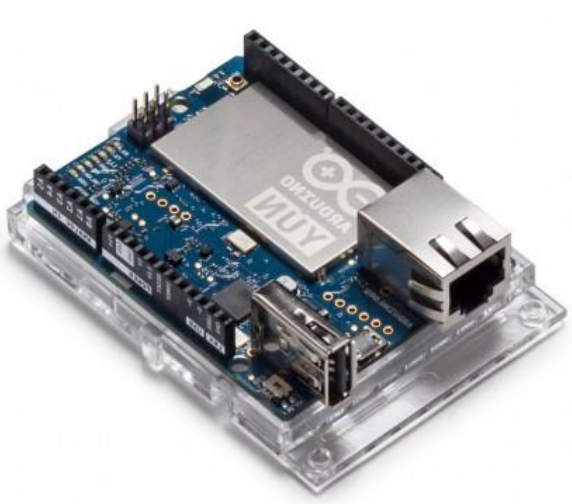


Figura 3-1. Arduino Yún. [1].

Arduino Yún es un microcontrolador basado en los procesadores ATmega32u4 y Atheros AR9331. El procesador Atheros soporta una distribución Linux basada en OpenWrt llamada Linino OS. La placa lleva integrados:

- Soporte para Ethernet y WiFi
- Un puerto USB, slot para tarjeta micro SD
- 20 pines de entrada/salida (7 de los cuales pueden ser usados como salidas PWM y 12 como entradas analógicas),
- Un oscilador de 16 MHz, una conexión micro USB,
- Entrada ICSP (In Chip Serial Programmer)
- 3 botones de Reset.

La diferencia del Yún respecto a otros dispositivos Arduino es su capacidad de comunicación con la distribución Linux de la placa, ofreciendo un poderoso computador conectado a la red con la sencillez de un Arduino. Además de Linux, ofrece la posibilidad de ejecutar scripts de Python para interacciones más robustas.

El Arduino Yún es similar a otras placas como el Arduino Leonardo, que también trabaja con el procesador ATmega32u4, excepto por la ejecución de Linux en la propia placa ya que no contiene un segundo procesador.

AVR Arduino microcontroller

Microcontroller	ATmega32U4
Operating Voltage	5V
Input Voltage	5 V
Digital I/O Pins	20
PWM Output	7
Analog I/O Pins	12
DC Current per I/O Pin	40 mA on I/O Pins; 50 mA on 3,3 Pin
Flash Memory	32 KB (of which 4 KB used by bootloader)
SRAM	2.5 KB
EEPROM	1 KB
Clock Speed	16 MHz

Figura 3-2. Características del procesador ATmega32u4. Arduino Yún.[Anexo A]

Arduino Microprocessor

Processor	Atheros AR9331
Architecture	MIPS
Operating Voltage	3.3V
Ethernet	802.3 10/100Mbit/s
WiFi	802.11b/g/n 2.4 GHz
USB Type	2.0 Host
Card Reader	Micro-SD
RAM	64 MB DDR2
Flash Memory	16 MB
SRAM	2.5 KB
EEPROM	1 KB
Clock Speed	400 MHz

Figura 3-3. Características del procesador Atheros AR9331. Arduino Yún. [Anexo A]

Este microcontrolador está especialmente pensado para aplicaciones IoT, ya que incorpora muchas opciones de conexión a internet y un procesador específico para ello, no obstante, no es el objeto de este proyecto el profundizar en estas características.

Simplemente comentar que durante el desarrollo de este proyecto se ha podido trabajar con la conexión WiFi de este dispositivo para cargar scripts y visualizar datos en el monitor serie de forma inalámbrica. Por lo que se entiende que en futuras versiones del dispositivo que se desarrolla en este proyecto se podrá trabajar de forma remota. Arduino Yún puede obtener datos de control mediante conexión WiFi, lo que abriría la puerta del mundo IoT a esta tecnología de grabación y control.

3.1.1 Pines, conexiones y periféricos

Este microcontrolador ofrece características parecidas al Arduino Uno en lo que a conexionado se refiere. Para este trabajo, destacar la ranura para tarjeta micro SD incorporada a la placa que ha sido una de las razones por la que se ha trabajado con esta placa, evitando así la necesidad de un modulo de hardware externo para este propósito.

Dispone 7 pines de entrada/salida digitales que permiten la configuración para trabajar con señales PWM, entre otros. También posee pines SDA y SCL habilitados para facilitar si conexión a un bus I2C. Esta característica será importante conforme avancemos en el desarrollo. A cotinuación se indican los diagramas de bloques y mapa de pines de Arduino Yún.

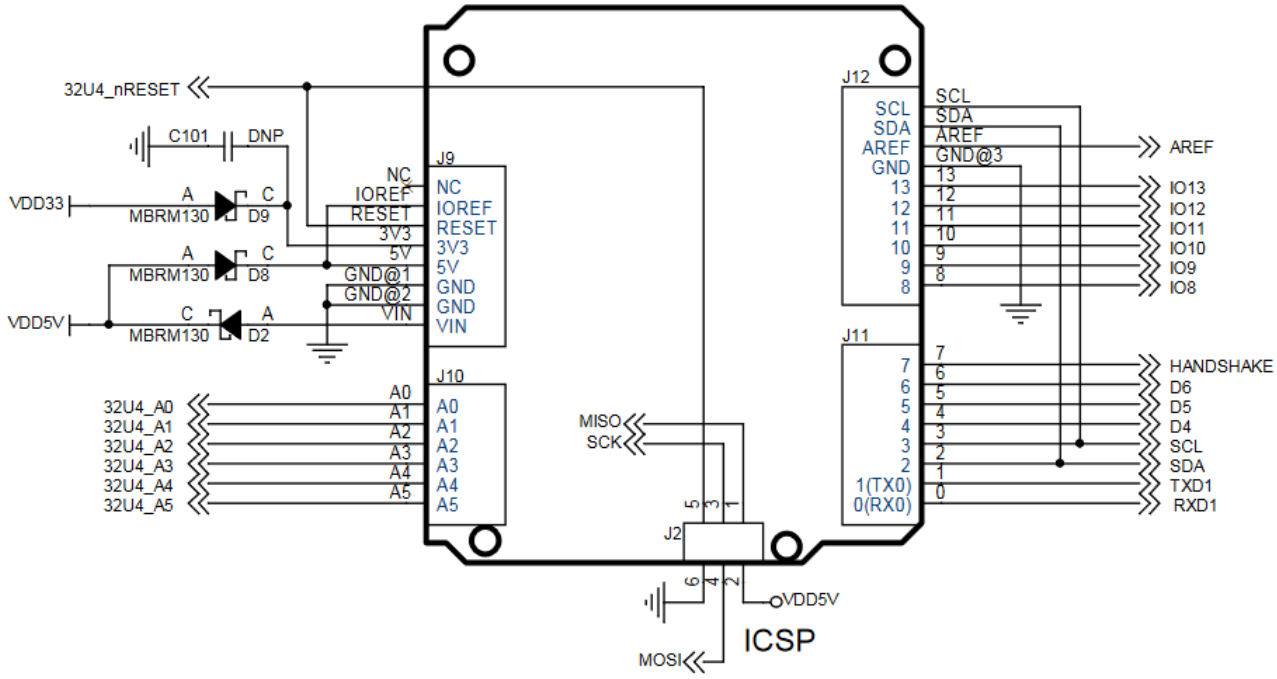


Figura 3-4. Mapa de pines de Arduino Yún. [Anexo A]

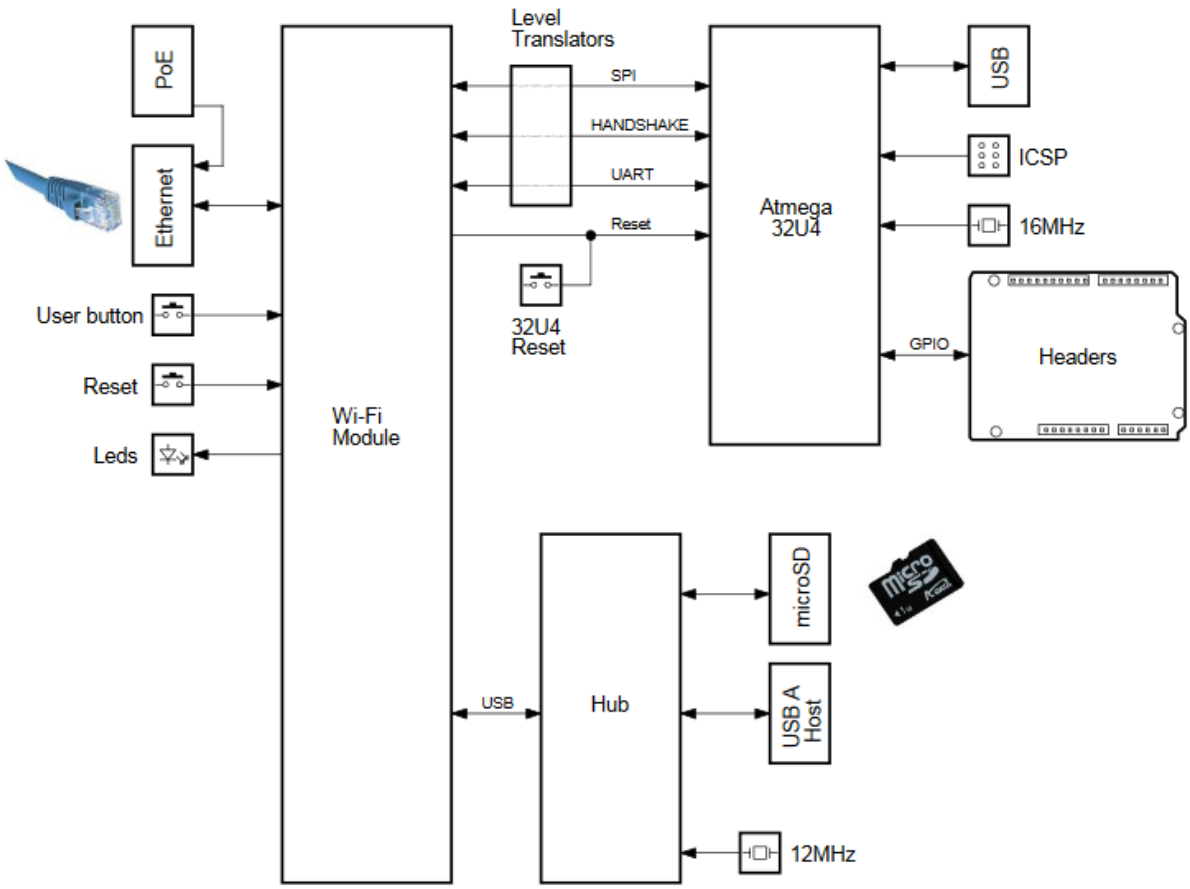


Figura 3-5. Dagrama de bloques principales de Arduino Yún. [Anexo A]

3.1.2 Librería *Bridge*

En el caso del Arduino Yún, a parte de las librerías comunes con el resto de dispositivos Arduino, cabe destacar la utilización de la librería *Bridge*.

Debido a su arquitectura, esta placa necesita esta librería para facilitar la comunicación entre los diferentes puertos, periféricos y procesadores.

Además de varias funciones propias de la librería, esta se divide en clases que están enfocadas a propósitos más concretos:

- Process: se usa para iniciar procesos en el procesador Linux y otras funciones relacionadas.
- Console: usada para comunicarse con el monitor de red. Funcionalmente muy parecida a la clase *Serial* que trabaja con el monitor serie.
- **FileIO:** Es la que se ha utilizado para este proyecto y se encarga de gestionar la lectura y escritura de archivos y datos en las memorias USB y SD.
- HttpClient: Crea clientes http en Linux.
- Mailbox: Interfaz asíncrona para comunicar Arduino y Linux.
- BridgeClient: Cliente HTTP basado en Arduino y modelado según la clase EthernetClient.
- BridgeServer: Servidor HTTP basado en Arduino y modelado según la clase EthernetClient.
- Temboo: Facilita la conexión a una gran variedad de herramientas online.

De todas estas clases, en este proyecto solo se ha trabajado con la clase *FileIO* para la gestión de lectura y escritura de datos en la tarjeta micro SD.

3.2. Arduino UNO

Figura 3-6. Arduino Uno [1].

Arduino UNO es una placa microcontroladora basada en el procesador ATmega328P. Lleva integrados los siguientes elementos:

- 14 pines digitales de entrada/salida (de los cuales 6 pueden ser utilizados como salidas PWM)
- 6 entradas analógicas
- Un oscilador de 16 MHz
- Una conexión USB
- Una entrada Jack de potencia
- Entrada ICSP (In Chip Serial Programmer)
- Un botón reset

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Figura 3-7. Características del procesador ATmega328P. Arduino UNO.[Anexo B]

3.2.1 Pines, conexiones y periféricos

En este caso, la placa Arduino Uno pertenece a la gama más estándar de Arduino, por lo que no está enfocada a aplicaciones concretas y pretende ser un dispositivo de propósito general y de fácil manejo. Esta es la razón de que muchas de las características, que ya se han visto en el Arduino Yún, se repitan en esta placa.

Destaca la opción de alimentar el dispositivo con una fuente de alimentación de 7 V – 12 V por la entrada Jack dispuesta en la placa para este propósito. Es una arquitectura simple y visual de cara a facilitar el rediseño por parte de los desarrolladores, atendiendo a esta nueva tendencia del hardware libre que ha prosperado gracias a este dispositivo y que se comentaba anteriormente.

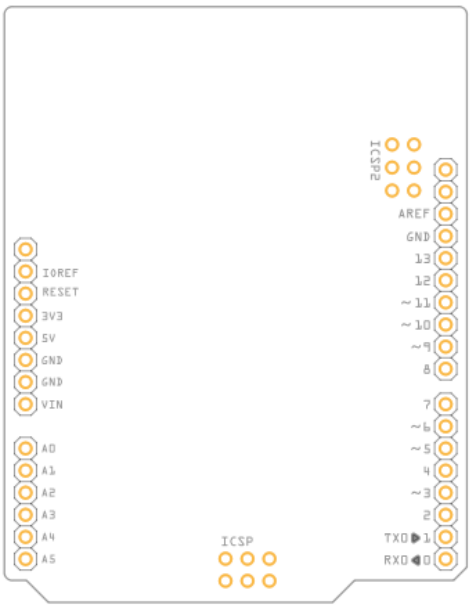


Figura 3-8. Mapa de pines de Arduino Uno. [2] .

3.3. Pantalla Táctil LCD TFT de 2.8 pulgadas

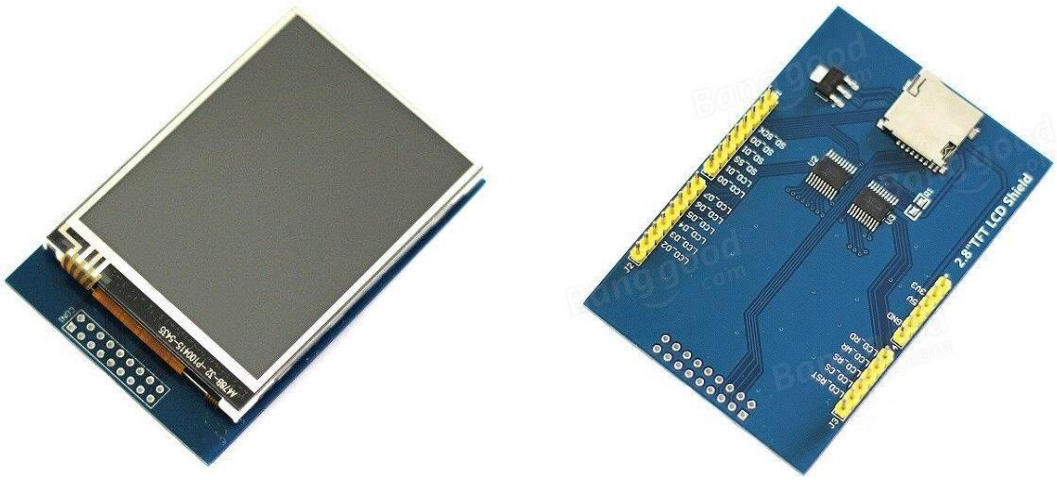


Figura 3-9. Pantalla Táctil LCD TFT de 2,8 pulgadas de Geekcreit. [3]

- Retroiluminación de 4 LED blancos
- Color de 18 bits 262,000 tonos diferentes
- Pantalla táctil resistiva de 4 hilos 240 x 320 píxeles con control individual de píxeles
- Pines dispuestos para ensamblar en Arduino UNO sin usar cableado y sin soldadura.
- Regulador 3.3V 300mA LDO integrado
- Compatible con alimentación a 5V compatible. Lógica de 3.3V o 5V
- Soporte Tarjeta de 2GB micro SD
- Tamaño: 7.8 x 5.3 cm

Esta pantalla táctil resistiva a color se ha utilizado para crear una interfaz en la que el usuario puede mandar órdenes básicas al dispositivo sin necesidad de un ordenador que ejecute los programas.

Los cuatro LED de retroiluminación hacen que el control del dispositivo no precise de una iluminación concreta para su uso, característica que a la hora de trabajar en cine, puede ser muy útil debido a los requerimientos de iluminación tan específicos que exigen ciertas escenas.

Permite crear interfaces dinámicas en las que tanto el texto como la imagen responden casi de inmediato al contacto con la pantalla.

Figura 3-10. Pruebas de concepto con la pantalla táctil.

El panel táctil es perfectamente compatible con librerías y funciones desarrolladas por la empresa Adafruit, cuya programación está ampliamente documentada en su web, destacando también el apartado *learn.adafruit* en el cual, tanto los usuarios como la propia marca, distribuye varios tutoriales y videos explicativos de la programación de sus dispositivos.

Los pines de esta pantalla han sido estratégicamente dispuestos para que cada señal que entra o sale de la pantalla por cada pin coincida con los pines de entrada y salida del modelo de placa Arduino Uno. Esta característica además de facilitar el montaje, permite un dispositivo bastante más compacto, menos pesado y un ahorro de cableado bastante significativo.

La rotulación de los pines facilita la identificación de las señales, lo cual permite, poder seleccionar pines para modificar señales y parámetros.

Esta pantalla también dispone de un slot para una tarjeta micro SD que permite cargar imágenes y logos de forma rápida y sencilla, y que conecta con el dispositivo Arduino UNO con la misma librería con la que se trabaja con otros shields SD.

Cabe destacar que Geekcreit imita arquitecturas de otros dispositivos y asume su software, de tal manera que hay ciertos aspectos en los que hay que prestar especial atención a la hora de adaptar la programación a la pantalla y repasar, y modificar, algunas librerías para hacer el dispositivo completamente funcional.

3.3.1 Librerías

Las librerías de Arduino y Adafruit, compatibles con este dispositivo, que se han utilizado para este proyecto son:

- <Adafruit_GFX.h>
- <TouchScreen.h>
- <Adafruit_TFTLCD.h>

3.4. Tarjeta micro SD

Hoy en día, debido a que la electrónica de consumo ha adoptado este dispositivo como método principal de almacenamiento extraíble junto con las memorias USB, existe una amplia gama de tarjetas y un rango enorme de posibilidades.

Los principales parámetros a tener en cuenta en este tipo de almacenamiento son la capacidad de almacenamiento, en cuyo caso, este proyecto no ha sido especialmente exigente, y también la velocidad de lectura y escritura, que ha sido uno de los parámetros más críticos a la hora de realizar el proyecto.

Debido a que el peso de los archivos .txt creados a partir de los datos de grabación no es nada significativo, en el caso de la capacidad de almacenamiento, cualquier tarjeta SD del mercado superior a 2 o 4 Gb supera con creces los requerimientos del proyecto en este aspecto.

Sin embargo, en lo que a velocidad de lectura y escritura se refiere, esta característica ha sido el principal limitante a la hora de sincronizar los procesos. Esto es consecuencia de tratar con una señal PPM con un periodo de 20 ms. Este periodo es bastante menor que el tiempo de lectura y escritura que precisa la tarjeta para poder obtener un dato.

Es por ello que la selección de la tarjeta SD ha atendido especialmente a la velocidad de lectura y escritura, siendo de esta uno de los parámetros más críticos de cara a la toma de decisiones en el proyecto. Los siguientes apartados describen los parámetros de selección de la tarjeta micro SD.

3.4.1 Tipo

El tipo de tarjeta está impreso en la misma y define la compatibilidad que tiene con los dispositivos. De esta forma una tarjeta micro SD solo será compatible con dispositivos de su mismo tipo o de tipos superiores en este orden: SDSC (Standard Capacity), SDHC (High Capacity) y SDXC (Extended Capacity).

Este proyecto utiliza una tarjeta micro SD de tipo HC.

Figura 3-11. Tipos de Tarjeta SD.

3.4.2 Capacidad

Depende de la capacidad del slot SD que trabajará con dicha tarjeta. En el caso de este proyecto se ha elegido una tarjeta de 32 Gb.

3.4.3 Velocidad de lectura /escritura

Se ha optado por una tarjeta micro SD de clase 10 lo cual quiere decir que su velocidad máxima de lectura escritura es de 10 Mb/s. De esta forma, a menos que el dispositivo Arduino en cuestión tenga capacidad de leer y escribir los datos más rápido, se asegura que las características de la tarjeta no limitan los tiempos.

3.4.4 Ultra High Speed (UHS)

Esta designación indica que esta tarjeta es compatible con dispositivos que admiten Ultra Alta Velocidad. Existen de clase U-1 y U-3. En el caso de nuestro proyecto es U-1.

3.4.5 Tarjeta SD seleccionada

- Tipo: HC
- Capacidad: 32 Gb
- Velocidad: Clase 10
- UHS: U-1

Figura 3-12. Tarjeta SD Samsung EVO 32 Gb. [4]

3.5. Emisora Eachine I6 de 6 canales

Existe una amplia variedad de emisoras de radiofrecuencia que se utilizan para multitud de aplicaciones y que ofrecen muchísimos parámetros configurables. En este caso se ha escogido la emisora Eachine i6 utilizada en aeromodelismo y control de drones.

Es una opción relativamente asequible que ha permitido desarrollar este proyecto sin complicar mucho la conexión y puesta a punto para obtener la señal deseada. A continuación se explican las características principales de esta emisora y aquellas que han influido de forma activa en el desarrollo del proyecto.

3.5.1 Esquema principal de la emisora RC

1. Asa
2. Antena
3. Interruptor A
4. Interruptor B
5. Interruptor C
6. Interruptor D
7. Stick izquierdo
8. Trim izquierdo 1
9. Trim derecho 1
10. Stick derecho
11. Trim izquierdo 2
12. Trim derecho 2
13. Arriba
14. Tecla Enter
15. Abajo
16. Cancelar
17. Sincronización
18. ON / OFF
19. Display LCD

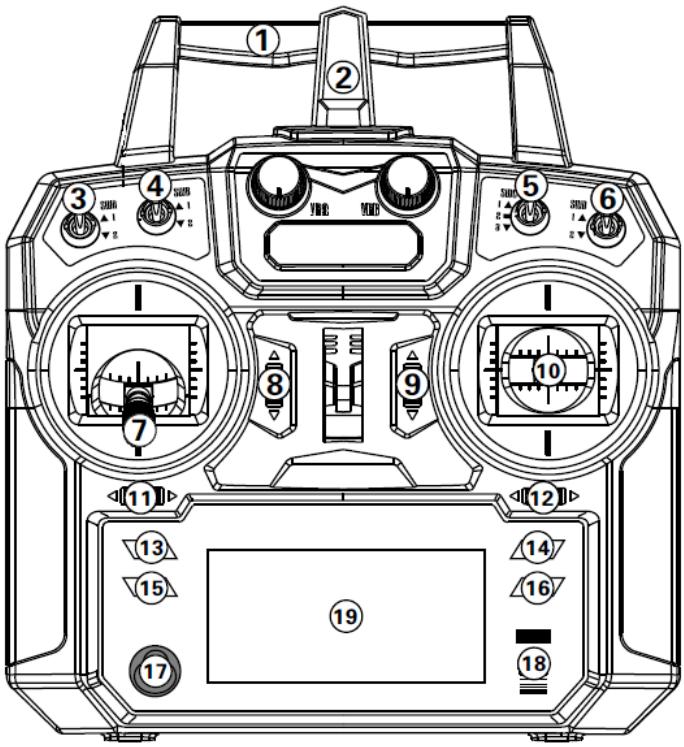


Figura 3-13. Esquema Eachine i6. [5]

3.5.2 Características principales

El fabricante Eachine ofrece en su web las características principales de los dispositivos que comercializa. En concreto esta emisora Eachine i6 siempre se vende junto con un drone de la marca, por lo que esta tabla de características se encuentra en cualquiera de los manuales entregados con los productos Eachine que la incorporan. Debido a esto, las características aquí expuestas están enfocadas al aeromodelismo y al vuelo de drones, por lo que algunos términos se refieren al propio vuelo en sí.

- Funciona en el rango de frecuencia de 2.405 a 2.475 GHz.
- Utiliza una antena multidireccional de alta ganancia y alta calidad.
- Cada emisora tiene una ID única
- Componentes electrónicos de baja potencia.
- El sistema AFHDS2A posee la función de identificación automática
- El sistema AFHDS2A incorpora codificación de canales múltiples y corrección de errores

Canales	6 Canales
Modelos	Glider / Heli / Airplane
Rango de RF	2.40-2.48GHz
Ancho de banda	500KHz
Banda	142
Potencia de RF	Menos de 20dBm
Sistema 2.4ghz	AFHDS 2A y AFHDS
Tipo de código	GFSK
Sensibilidad	1024
Advertencia de bajo voltaje	Menos de 4.2V
Puerto DSC	PS2
Salida	PPM
Puerto de cargador	No
longitud de antena	26mm (doble antena)
Peso	392g
Potencia	6V 1.5AA * 4
Modo de pantalla	Transflectiva STN positiva, matriz de puntos 128 * 64 VA73 * 39 mm, luz de fondo blanca.
Tamaño	174 x 89 x 190 mm
Actualización en línea	Sí
Color	Negro
Certificado	CE0678, FCC
Número de memorias de modelos	20
Orden de canales	Aleron – CH1 Elevator – CH2 Throttle – CH3 Rudder – CH4 Reserva – CH5 Reserva – CH

Tabla 3-1. Hoja de características del Eachine i6. [6]

3.5.3 Salida PPM y configuración

Una vez revisadas las principales características de la emisora se van a repasar las características que intervienen en este proyecto, en concreto las que tienen que ver con la generación de señales PPM.

Para simplificar el problema, a nivel conceptual, la emisora se trata como una fuente de señales PPM sin entrar en los parámetros físicos de la señal. Tan solo se tendrán en cuenta los parámetros temporales y la lógica de la señal.

A la espalda de la emisora hay un puerto de salida S-Video pensado para la interconexión de la emisora con un ordenador usando el modo training que permite usar la emisora en simuladores de vuelo. Sin embargo, nuestra aplicación solo requiere conectar el dispositivo a la salida PPM y a la masa de la emisora para referenciar la señal. De esta forma se conectan cables unipolares solo a estas dos señales.

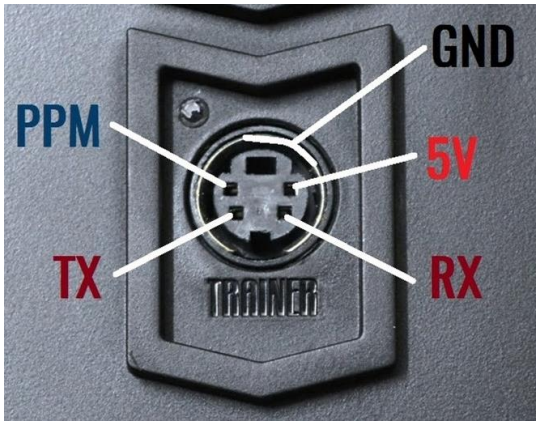


Figura 3-14. Puerto de salida Eachine i6.

Figura 3-15. Conexionado a la salida Eachine i6.

En principio, al encender la emisora no se recibe la señal por el puerto S-Video. Es necesario acceder al menú de la emisora *System*, y activar el *Student Mode* en la pestaña de ajustes.

Para acceder al menú hay que encender la emisora con los botones *Trim izquierdo* 2 presionado hacia la derecha y *Trim derecho* 2 presionado hacia la izquierda. Una vez hecho esto se pulsa la tecla *Enter* hasta que aparezca el menú.

Figura 3-16. Menú del Eachine i6.

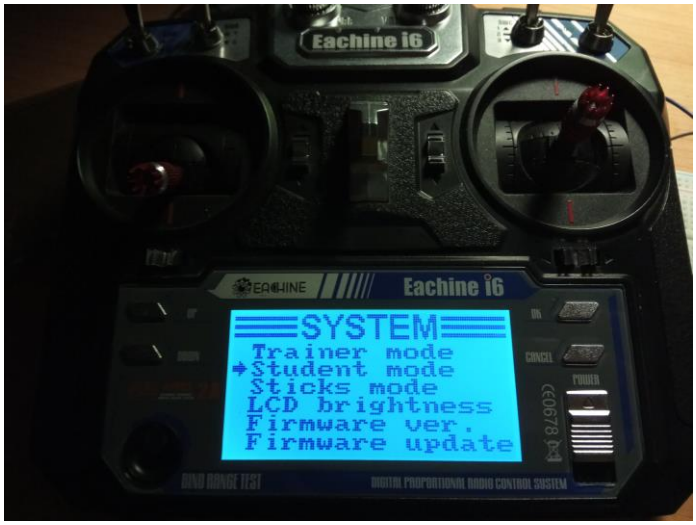


Figura 3-17. Ajustes de sistema del Eachine i6.

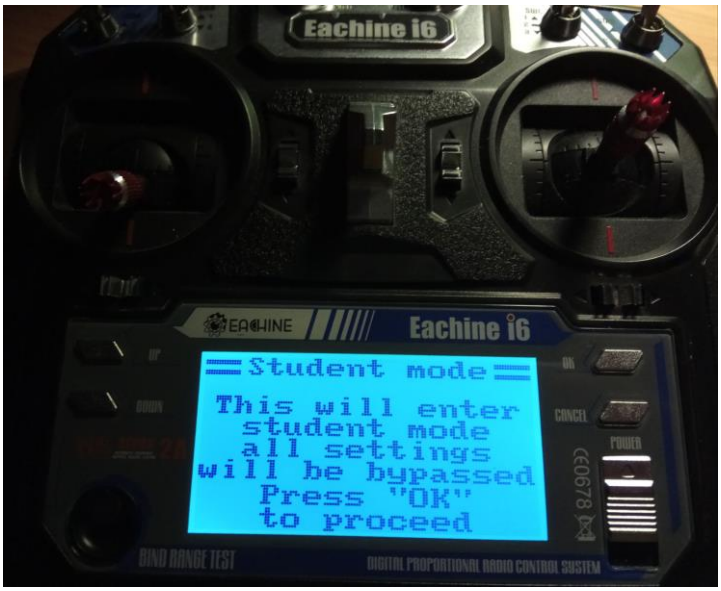


Figura 3-18. Student Mode del Eachine i6.

Esta operación debe repetirse cada vez que se enciende la emisora. Este modo esta diseñado para conectar la emisora al ordenador y poder entrenar en simuladores de vuelo. En nuestro caso, obtenemos del pin indicado la señal de PPM de salida para empezar a trabajar con ella.

Antes de realizar ninguna prueba se han identificado los parámetros de la señal, para facilitar luego la caracterización de la misma. Tras comprobar la señal con un analizador de señales, se obtiene una gráfica parecida a la que se muestra a continuación:

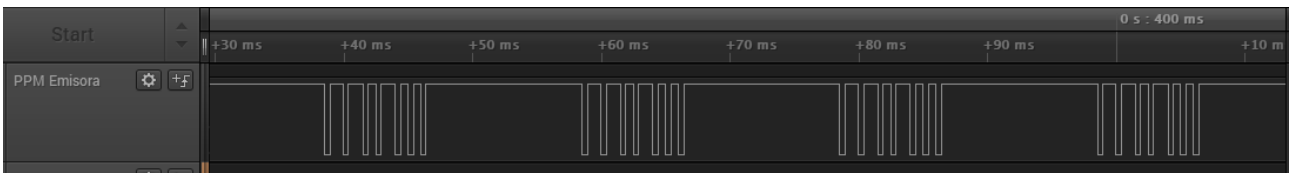


Figura 3-19. Salida PPM del Eachine i6.

Se observa que la señal emitida por el mando corresponde a los siguientes parámetros:

- Periodo: 20 ms
- Canales: 6
- Tipo de señal: Negativa o Inversa
- Gaps entre pulsos: 400 μs

3.5.4 Canales

Como se comentó anteriormente la señal PPM se compone de canales que contienen los datos necesarios para el control. En este caso la señal emitida tiene 6 canales, aunque esta emisora puede ser modificada para utilizar hasta 10 canales. No obstante, este proyecto no requiere tales características. Simplemente se trata de identificar en qué eje de cada palanca y en qué interruptores se controla el ancho de los pulsos de cada canal.

Navegando de nuevo por el menú de la emisora, se llega a la siguiente pantalla donde se identifican los canales que se manejan a través de los ejes horizontales y verticales de las palancas de la emisora.

Figura 3-20. Configuración de canales 1, 2, 3 y 4. Eachine i6.

Para encontrar los canales que faltan solo hay que dirigirse a la pestaña de canales auxiliares donde se especifica la ubicación de los dispositivos que actúan sobre el resto de canales.

Estos controles son completamente configurables, es decir, se puede modificar qué canal se controla con qué dispositivo para tener un control adaptado a las diferentes aplicaciones. Cabe recordar que, aunque esta emisora esté enfocada al vuelo de aeromodelisto, el propósito de este proyecto es el de controlar todo tipo de robots animatrónicos, con lo que esta opción facilita mucho el trabajo de cara al montaje de este tipo de prototipos.

Figura 3-21. Configuración de canales 5 y 6 en interruptores de 2 posiciones. Eachine i6.

Figura 3-22. Interruptores de 2 posiciones SWA y SWB. Eachine i6.

3.6. Analizador lógico

Este elemento hardware empezó siendo una herramienta auxiliar pero se ha convertido en una parte prácticamente indispensable del hardware empleado en el desarrollo de este proyecto.

El análisis gráfico de las señales se ha convertido en la forma más sencilla y clara de ver lo que realmente ha ocurrido en los dispositivos y de qué forma ha ido avanzando el proyecto. A nivel de investigación lo más lógico y exacto habría sido utilizar un osciloscopio para realizar aproximaciones más exactas a la señal. Pero esta opción ha funcionado sorprendentemente bien y ha cumplido con creces los objetivos del alcance de este proyecto.

Este analizador ha sido utilizado con el software *Saleae Logic 1.2.14* del que se han extraido todas las gráficas de señales mostradas en esta memoria.

Los pines de entrada reciben las señales a analizar y la salida usb envía los datos necesarios al ordenador para crear las gráficas con el software de análisis.

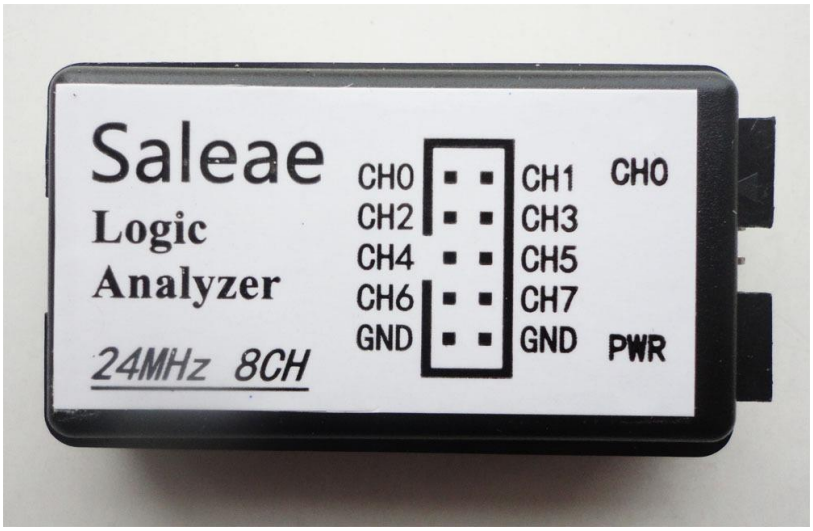


Figura 3-23. Analizador lógico.

Sus características más significativas son las siguientes:

- Salida: USB.
- Entrada: 10 pines dispuestos en una matriz de 5 x 2.
- 2 Pines GND de referencia.
- 8 canales de entrada.
- 24 MHz de frecuencia de muestreo.

El analizador lógico capta señales a nivel alto y bajo y los flancos de subida y bajada de las señales lógicas, incluyendo periodos y todo tipo de intervalos de tiempo. Sin embargo, esta tecnología no mide niveles de tensión e intensidad, y no esta preparado para soportar grandes valores de tensión o intensidad.

No obstante, junto con el software adecuado y un ordenador, ha sido la herramienta utilizada para visualizar las señales y comprender mejor el problema.

4 ESTUDIO DE LA SEÑAL Y ALTERNATIVAS DE SOLUCIÓN CONTEMPLADAS

El desarrollo de la solución se ha ido planteando de forma que los pasos posteriores han ido completando la solución previa y descartando opciones que no han sido viables a la hora de la implementación. Esto ha dado como resutado la comprobación de varias alternativas de diseño hasta llegar al prototipo que mejor se ha adaptado a los requisitos del proyecto.

Muchas de las técnicas estudiadas no se habían considerado a priori al inicio del proyecto, sin embargo, la necesidad de dar ciertos pasos intermedios ha proporcionado un conocimiento más amplio y profundo del problema y de la aplicación final.

4.1 Tratamiento y caracterización de la señal

Antes focalizar el trabajo en una aplicación concreta ha sido necesario extraer los datos de la señal de entrada de forma clara y estable. De esta forma se ha conseguido aislar cada dato por cada canal y poder trabajar con ellos más fácilmente

4.1.1 Medición de los pulsos de la señal de entrada

Tras conocer y estudiar otros proyectos en los que se captan este tipo de señales se concluye que la mejor forma de caracterizar una señal PPM es almacenar la duración de los pulsos en cada canal expresados en microsegundos.

Estos valores podrán ser almacenados en arrays dimensionados según el número de canales. La emisora utilizada trabaja con 6 canales. Se añade una posición más para contemplar los espacios entre tramas. También se define un vector donde se almacenan los instantes inicial y final del pulso.

```
#define ncanales 6
unsigned int senal[ncanales+1];
tiempo[2];
```

Se define una interrupción por hardware cada flanco de bajada donde se guarda el instante de tiempo en la variable *t* utilizando la función *micros()*, que devuelve el instante absoluto desde que se inició el programa expresado en microsegundos:

```
void bajada(){
  t=micros();                 //Interrupción flanco de bajada
}
```

En este, sentido cabe destacar que no es necesario tener en cuenta los flancos de subida de la señal ya que los gaps de sincronización forman parte de la duración del pulso. Por lo tanto solo se tendrán en cuenta los flancos de bajada en el caso de esta aplicación.

Figura 4-1. Captura de analizador lógico: Pulsos entre flancos de bajada.

Durante la ejecución en bucle del programa en Arduino se utiliza el vector *tiempo* para almacenar los instantes actual y previo. Cada vez que entra un valor nuevo se actualiza el vector.

Los dos valores almacenados en el vector se restan para obtener el intervalo de tiempo entre los dos últimos flancos de bajada, con lo que se obtiene el valor *interv*.

Este valor *interv* se almacena en el vector *senal* y la variable *canal* indica el lugar en el que debe ser almacenado este intervalo de tiempo medido. Es decir, se posicionan los datos en el vector de la misma manera en la que vienen dados en la señal.

```
if (t != tiempo[0]) {           //Si tiempo[0] es distinto de t
  tiempo[1]=tiempo[0];          //Se corre valor hacia tiempo[1]
  tiempo[0]=t;                  //Se almacena nuevo valor t
  canal++;                      //Actualizar valor del canal
}

interv = tiempo[0]-tiempo[1];   //Intervalo de tiempo entre flancos de bajada
senal[canal]=interv;            //Duración del intervalo en el canal correspondiente
```

El vector *senal* una vez completo, tendrá almacenados todos los valores temporales de una trama completa, incluyendo el canal 0, que caracteriza el tiempo vacío de sicronización entre trama y trama.

4.1.2 Obtención de tramas PPM

Una vez obtenidos los pulsos se utiliza el siguiente código para imprimir por pantalla el vector junto al canal correspondiente. Se obtienen los valores de cada trama mostrados como aparecen en la figura 4-2.

```
#define ncanales 6
unsigned int senal[ncanales+1], tiempo[2];
int canalsig, interv=0, canal=0; //Canal siguiente, intervalo de tiempo y canal actual
int t;                           //Instante de tiempo

void setup() {
  Serial.begin(2000000);
  pinMode(7,INPUT_PULLUP);     //Entrada señal PPM para interrupción flanco de bajada
  attachInterrupt(4,bajada,FALLING);    //Interrupción flanco de bajada por el pin 7
}
void loop() {
canalsig=canal+1;
if (t != tiempo[0]) {           //Si tiempo[0] es distinto de t
  tiempo[1]=tiempo[0];          //Se corre valor hacia tiempo[1]
  tiempo[0]=t;                  //Se almacena nuevo valor t
  canal++;                      //Actualizar valor del canal
}
interv = tiempo[0]-tiempo[1];   //Intervalo de tiempo entre flancos de bajada
senal[canal]=interv;            //Duración del intervalo en el canal correspondiente

if (canal == canalsig){         //Al pasar al canal siguiente
  Serial.print (senal[canal-1]);
  Serial.print (“/”);
  Serial.println (canal-1);
  }
if (canal > ncanales) {          //Limitación por canales
  canal = 0;
  }
if (pulseIn(7, HIGH) > 3000){     //Sincronización
  canal = 0;
  }
}
void bajada(){                    //Interrupción flanco de bajada
  t=micros();
}
```

Estos valores serán mostrados como sigue, incluyendo el valor en microsegundos del dato y el canal que ocupan separados por una barra.

```
12516/0
1464/1
1500/2
1008/3
1512/4
1000/5
1000/6
```

Figura 4-2. Trama completa impresa por monitor serie de arduino.

4.1.3 Generación de tramas PPM por un pin arduino

Una vez obtenidos los datos de una trama se va a definir una única trama tipo para realizar pruebas dando el paso inverso y generando una señal PPM desde un pin digital del Arduino Yún.

En este caso se usará la función *delayMicroseconds()* que interrumpe la ejecución del programa durante tantos microsegundos como valor contengan los paréntesis.

```
digitalWrite(4, LOW);
delayMicroseconds(400);
digitalWrite(4, HIGH);
delayMicroseconds(senal[0]-400);
digitalWrite(4, LOW);
delayMicroseconds(400);
digitalWrite(4, HIGH);
delayMicroseconds(senal[1]-400);
digitalWrite(4, LOW);
delayMicroseconds(400);
digitalWrite(4, HIGH);
delayMicroseconds(senal[2]-400);
digitalWrite(4, LOW);
delayMicroseconds(400);
digitalWrite(4, HIGH);
delayMicroseconds(senal[3]-400);
digitalWrite(4, LOW);
delayMicroseconds(400);
digitalWrite(4, HIGH);
delayMicroseconds(ppm[4]-400);
digitalWrite(4, LOW);
delayMicroseconds(400);
digitalWrite(4, HIGH);
delayMicroseconds(ppm[5]-400);
digitalWrite(4, LOW);
delayMicroseconds(400);
digitalWrite(4, HIGH);
delayMicroseconds(ppm[6]-400);
```

Se van a ir intercalando niveles bajos y altos en el pin de salida trabajando con los invervalos de tiempo obtenidos anteriormente para repetir esta señal.

De esta forma se consigue repetir tramas PPM con los valores deseados por uno de los pines digitales del Arduino Yún.

4.2 Implementación en Arduino Yún

Una vez obtenidos los datos de la señal PPM de forma estable, se pasa a la implementación en Arduino Yún de las funciones de grabación y reproducción en la tarjeta SD.

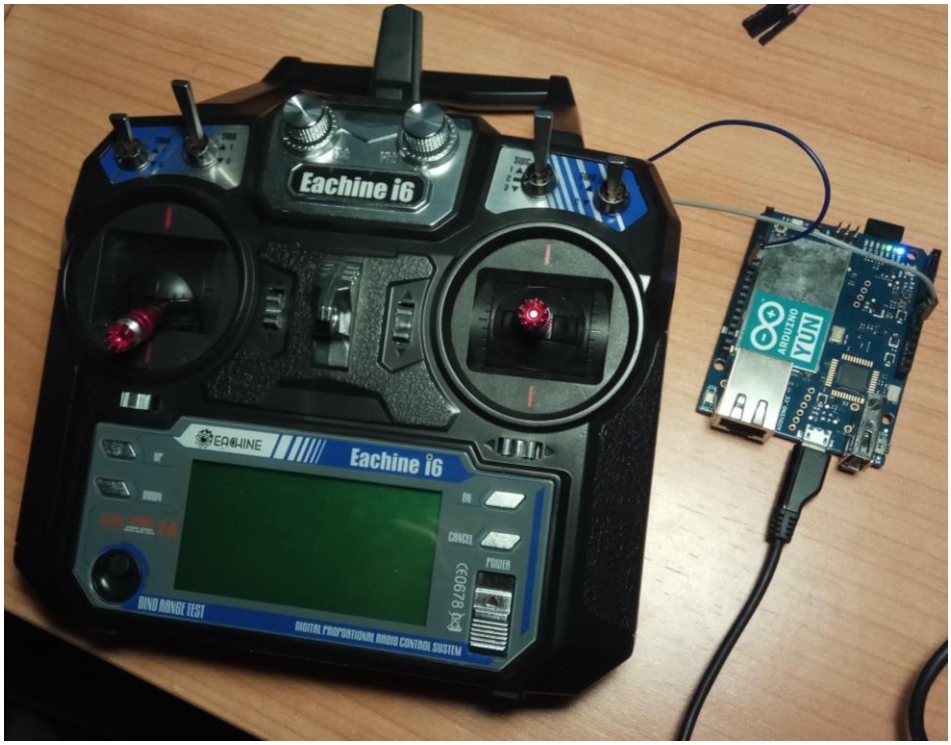


Figura 4-3. Implementación del sistema en Arduino Yún.

4.2.1 Librerías de lectura y escritura en la tarjeta SD

Como se comentó anteriormente, Arduino Yún posee unas librerías muy concretas para gestionar la lectura y escritura de datos en la tarjeta SD debido a su arquitectura interna de dos procesadores. Teniendo esto en cuenta, se ha desarrollado un código basado en la librería *Bridge* utilizando las funciones de la clase *FileIO*.

```
#include <FileIO.h>      //funciones relacionadas con la lectura y escritura de la SD
```

```
void setup() {
  Bridge.begin();       //Inicializar librería bridge para comunicar con Atheros 9331
  FileSystem.begin();   //Gestor de archivos de Bridge

  (otras inicializaciones...)
}
```

Se decide almacenar los datos obtenidos como cadena de caracteres, para mantener el formato de visualización de los datos junto con el canal correspondiente y facilitar así, tanto la lectura del archivo como la comprensión de los datos almacenados.

```
12516/0
1464/1
1500/2
1008/3
1512/4
1000/5
1000/6
```

Figura 4-4. Formato de trama completa almacenada en la tarjeta SD.

4.2.2 Escritura en la tarjeta SD

Al realizar los primeros intentos de escritura de los datos sobre la tarjeta SD se ha observado que los valores impresos en el monitor serie de arduino no están dentro del margen de operación normal de la señal PPM, obteniendo valores de los intervalos de tiempo muy diferentes a los deseados. También se ha observado que no todos los canales de la trama eran captados.

En primer lugar, para ver como se están captando los datos se han eliminado las funciones de escritura de la tarjeta SD en el mismo script y los datos volvían a obtenerse correctamente, por lo que se intuyó que el proceso de escritura interfería con el cálculo de la duración de los pulsos.

Para clarificar esta situación se ha procedido al análisis gráfico de la señal estableciendo un pulso de comprobación a nivel bajo por defecto, y a nivel alto durante el proceso de escritura.

También se ha trabajado con interrupciones por flanco de subida para ver si Arduino sigue siendo capaz de seguir la señal PPM cuando se le introducen estas nuevas funciones de gestión de la tarjeta SD.

```
void setup() {
  Bridge.begin();       //Inicializar librería bridge para comunicar con Atheros 9331
  FileSystem.begin();   //Gestor de archivos de Bridge

  Serial.begin(2000000);
  pinMode(7,INPUT_PULLUP);     //Entrada señal PPM para interrupción flanco de bajada
  pinMode(2,INPUT_PULLUP);     //Entrada señal PPM para interrupción flanco de subida
  pinMode(4,OUTPUT);                    //Salida de la señal para comprobaciones
  pinMode(3,OUTPUT);                    //Salida de la señal para comprobaciones
  attachInterrupt(4,bajada,FALLING);    //Interrupción flanco de bajada por el pin 7
  attachInterrupt(1,subida,RISING);     //Interrupción flanco de subida por el pin 2
}
```

Observando estas señales de forma gráfica se obtiene el resultado mostrado a continuación.

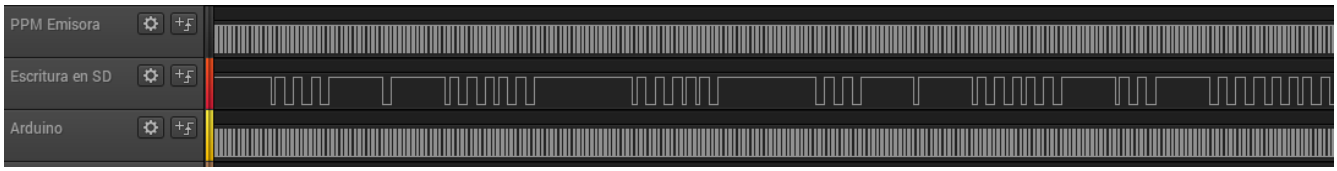


Figura 4-5. Captura de analizador lógico: Escritura en SD y seguimiento de Arduino, frente a PPM.

Se observan intervalos de tiempo de escritura irregulares, y más largos que el periodo de la trama. Esto se debe a que el tiempo de acceso a la tarjeta micro SD excede con creces el periodo de la señal.

Para intentar solucionar este problema se ha establecido una variable *flag* que habilita o interrumpe el cálculo de los pulsos mientras se realiza la escritura en la tarjeta SD, obteniendo de nuevo, datos correctos.

```
#define ncanales 6
#include <FileIO.h>

unsigned int senal[ncanales+1], tiempo[2];
int canalsig, interv=0, canal=0;
int t, flag;                            //Instante de tiempo y Bandera para grabacion
String trama, canal_str, senal_str;     //Cadenas de trama, canal y pulso de señal

void setup() {
  Bridge.begin();       //Inicializar librería bridge para comunicar con Atheros 9331
  FileSystem.begin();   //Gestor de archivos de Bridge

  Serial.begin(2000000);
  pinMode(7,INPUT_PULLUP);           //Entrada PPM para interrupción flanco de bajada
  pinMode(2,INPUT_PULLUP);           //Entrada PPM para interrupción flanco de subida
  pinMode(4,OUTPUT);                 //Salida de la señal para comprobaciones
  pinMode(3,OUTPUT);                 //Salida de la señal para comprobaciones
  attachInterrupt(4,bajada,FALLING); //Interrupción flanco de bajada por el pin 7
  attachInterrupt(1,subida,RISING);  //Interrupción flanco de subida por el pin 2
}
```

```
void loop() {
canalsig=canal+1;
if (t != tiempo[0]) {           //Si tiempo[0] es distinto de t
  tiempo[1]=tiempo[0];          //Se corre valor hacia tiempo[1]
  tiempo[0]=t;                  //Se almacena nuevo valor t
  canal++;                      //Actualizar valor del canal
}

interv = tiempo[0]-tiempo[1];   //Intervalo de tiempo entre flancos de bajada
senal[canal]=interv;            //Duración del intervalo en el canal correspondiente

if (canal == canalsig){             //Si el canal aumenta
  canal_str = String(canal-1);      //Se pasan los parámetros a cadena de caracteres
  senal_str = String(senal[canal]);
  if (flag==0){                     //Y si no se está guardando nada en este instante
     trama += (senal_str + "/" + canal_str + '\r' + '\n');
  }
}
if (canal > ncanales) {            //Limitación por canales
  canal = 0;
  Serial.print(trama);
  digitalWrite(3, HIGH);        //pulso de comprobación INICIO DE GUARDADO
  File dataFile = FileSystem.open("/mnt/sd/grabacion008.txt", FILE_APPEND);
  flag=1;
  if (dataFile) {
    dataFile.print(trama);
    dataFile.close();
    flag=0;
    digitalWrite(3, LOW);         //pulso de comprobación FINAL DE GUARDADO
    }
  trama ="";
}
if (pulseIn(7, HIGH) > 3000){               //Sincronización
  canal = 0;
  trama="";
}
}
```

```
void bajada(){
  t=micros();                 //Interrupción flanco de bajada
  digitalWrite(4, HIGH);      //pulso de comprobación SEGUIMIENTO DE LA SEÑAL
}

void subida(){                //Interrupción flanco de subida
  digitalWrite(4, LOW);       //pulso de comprobación SEGUIMIENTO DE LA SEÑAL
}
```

Trabajando con el analizador lógico se obtiene finalmente esta gráfica que muestra lo que realmente está pasando:

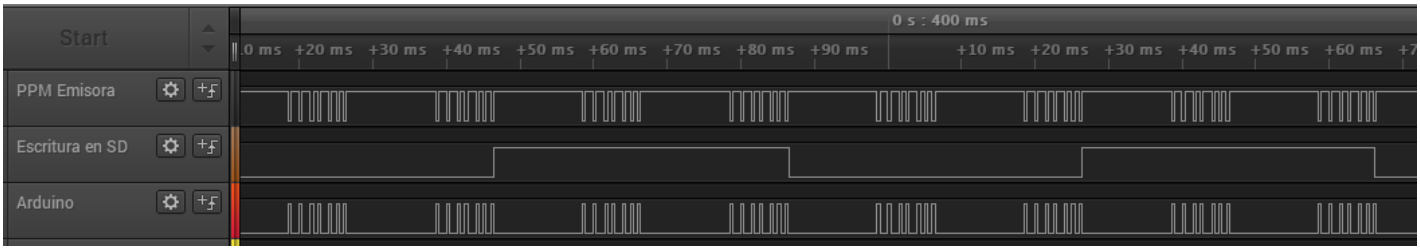


Figura 4-6. Captura de analizador lógico: Escritura en SD y seguimiento de Arduino, frente a PPM.

La gráfica obtenida muestra como Arduino no tiene problema de seguimiento de la señal. Sin embargo, el proceso de escritura de una trama se prolonga durante dos ciclos de la señal e ignora la tercera trama. Es la cuarta trama la que vuelve a ser escrita en la tarjeta SD.

Por esta razón se toma, como solución a la grabación de la señal, la escritura de 1 de cada 4 tramas recibidas. Este muestreo permite almacenar sin problemas los datos de la señal de control obteniendo datos de forma correcta y asumiendo cierta pérdida de información.

En la reproducción de la señal se tendrá en cuenta esta característica, y cada trama leida se reproducirá 4 veces, para volver a obtener una señal en las mismas condiciones en las que se grabó.

Sabiendo que se envían 50 tramas cada segundo, este muestreo de la señal se considera una aproximación válida que se pretende respaldar mas adelante con análisis de los datos y pruebas físicas una vez se reproduzca la señal previamente grabada.

4.2.3 Lectura de la tarjeta SD

Una vez obtenido un archivo *.txt* con las tramas recibidas guardadas en la tarjeta SD, es el momento de plantear la lectura de esos archivos para su posterior reproducción por uno de los pines del Arduino Yún.

Como se vió anteriormente para generar las tramas ppm por un pin de salida es necesario leer los datos de una trama entera. En este aspecto se vuelve a encontrar el mismo problema que en la escritura de datos.

Debido al tiempo de carga de los datos, no se puede trabajar en paralelo con la generación de señales por un pin de salida mientras se leen los datos desde la SD. La lectura de un dato tarda bastante más que su reproducción en tiempo real.

Se opta por cargar los datos previamente y reproducir 4 veces cada trama para compensar el muestreo realizado cuando se ha grabado la señal.

Arduino no tiene mucha memoria interna. Debido a esto se ha tenido que crear una matriz *ppm[i][ncanales+1]* cuyas filas son cada una de las tramas y cada columna es un canal de la señal. Se define en número máximo de filas de la matriz con la constante *ntramas* para no cargar más tramas de las que Arduino puede ejecutar de forma estable. Una vez cargados estos datos comienza la reproducción de la señal.

```
#define ncanales 6
#include <FileIO.h>
#define ntramas 120                     //Numero máximo de tramas cargadas
unsigned int ppm[ntramas][ncanales+1];  //Buffer matricial
boolean leido=false;                    //Variable para indicar el fin de la carga
unsigned int can, i=0, j=0, pulso;      //Variables para trabajar con la lectura

void setup() {
  Bridge.begin();
  Serial.begin(9600);
  FileSystem.begin();
  pinMode(4,OUTPUT);                    //Pin de salida de la señal PPM
  digitalWrite(4, HIGH);                //PPM a nivel alto por defecto (negativa)
}

void loop () {
char c;
String s;
//Lectura del archivo de texto
File myFile = FileSystem.open("/mnt/sd/grabacion009.txt", FILE_READ);

  if (myFile && !leido) {           //Si el archivo no se ha leído y esta disponible
      while (myFile.available())
      {
          c=myFile.read();          //Lectura de un caracter
          s=s+String(c);            //Incluir carácter en cadena de caracteres
          if ( c=='/'){             //Si el carácter es una barra
              s.replace("/", "");   //Eliminarlo de la cadena previa
              pulso=s.toInt();      //Pasar la cadena a entero
              s="";                 //Vacíar la cadena de caracteres
                  }
```

```
         else if ( c=='\n'){             //Si hay un salto de línea
             s.replace("\r", "");        //Eliminarlo de la cadena previa
             can=s.toInt();              //Almacenar el carácter previo como canal
             s="";                       //Vaciar la cadena de caracteres
             ppm[i][can]=pulso;          //Incluir el dato pulso en el buffer
             }
      if (can == 6 ){                    //Visualización de la carga de datos
             Serial.print(ppm[i][0]);
             Serial.print(";");
             Serial.print(ppm[i][1]);
             Serial.print(";");
             Serial.print(ppm[i][2]);
             //(...)                     //Se imprime la matriz entera
             can=0;
             i++;
         }
       }
          leido=true; }            //Si el archivo deja de estar disponible, leído.

else if (leido){                   //Si se ha acabado de leer, se cierra el archivo
         myFile.close();
         Serial.println("leido");

      while (j<=ntramas){
           digitalWrite(4, LOW);                //primera repetición de 4
           delayMicroseconds(400);
           digitalWrite(4, HIGH);
           delayMicroseconds(ppm[j][0]-400);
           digitalWrite(4, LOW);
           delayMicroseconds(400);
           digitalWrite(4, HIGH);
           delayMicroseconds(ppm[j][1]-400);
           digitalWrite(4, LOW);
           delayMicroseconds(400);
           //(...)                    //se continua hasta completar la trama
           //(...x4)                  //se repite la misma trama 4 veces
           j++; }                     //pasa a leer la siguiente fila de la matriz
                                      //y repite el proceso hasta que j==ntramas
```

```
            Serial.println("Hecho!");          //Reproducción finalizada
            digitalWrite(4,HIGH);
            delay(60000);
  }
  else {
    Serial.println("error abriendo el archivo.txt");
  }
}
```

Esta solución además de ineficiente solo proporciona unos 9 segundos de reproducción, limitados por la capacidad del buffer cargado, definido por la constante *ntramas*, en la memoria interna de Arduino. No obstante, se puede decir que se ha conseguido grabar y reproducir una señal PPM con esta plataforma.

4.2.4 Solución provisional

Se ha obtenido un dispositivo basado en Arduino Yún que es capaz de muestrear la señal PPM recibida por uno de sus pines digitales y almacenarla como cadena de caracteres en un archivo *.txt* en una tarjeta micro SD.

Este dispositivo es también capaz de generar una señal PPM por uno de sus pines digitales, leyendo un archivo *.txt* y compensando el muestreo realizado en la grabación. La señal generada tiene una duración máxima de 9,60 segundos.

4.2.5 Conclusiones

Se consigue grabar y reproducir una señal de control con una plataforma Arduino. En los siguientes prototipos se intentará mejorar la calidad de la grabación y de reproducción e integrar esta tecnología.

Se intentará reducir la pérdida de información en el muestreo de la grabación, y también se intentará prolongar el tiempo de reproducción optimizando el código.

4.3 Implementación en Arduino Yún + Arduino Uno

En el anterior prototipo uno de los principales problemas ha sido la velocidad de lectura y escritura de la tarjeta micro SD y la dificultad de Arduino al realizar varias tareas al mismo tiempo sin una buena sincronización.

Se pretende mejorar las prestaciones del dispositivo en desarrollo añadiendo una placa de Arduino Uno al montaje y utilizando una pantalla táctil para obtener una interfaz gráfica que facilite el manejo de las funciones de grabación y reproducción.

Se pretende sincronizar ambas placas para que mientras que una capta los datos de la señal PPM otra pueda estar guardando los archivos en la SD al mismo tiempo.

También se ha diseñado una interfaz de usuario en la pantalla táctil que permite envíar ordenes a ambos dispositivos con un solo tooque de pantalla, sin necesidad de duplicar la orden a realizar.

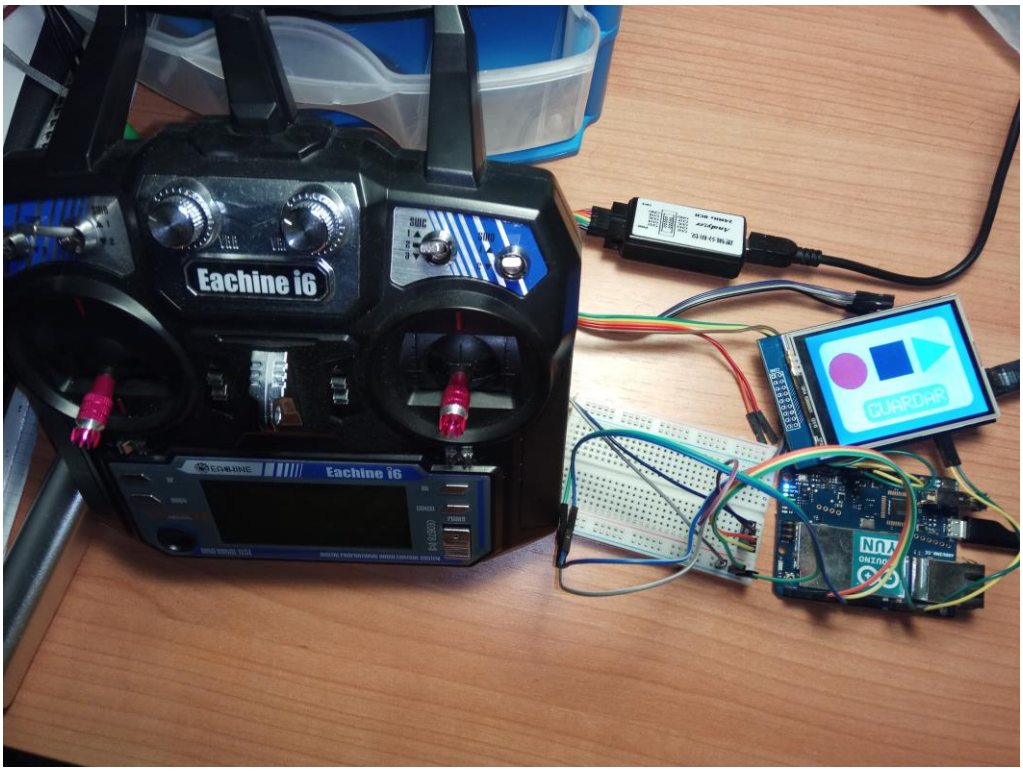


Figura 4-7. Implementación del sistema en Arduino Yún + Arduino Uno.

4.3.1 Comunicaciones

Al trabajar con más de un microcontrolador ha sido necesario investigar la mejor forma de comunicación entre ellos, atendiendo a las necesidades de transmisión de datos y gestión de la tarjeta SD. También se ha tenido en cuenta que el módulo de la pantalla táctil también ha sido un dispositivo a comunicar, aunque en este caso las librerías de este módulo gráfico han facilitado la programación.

En arduino se trabaja con muchos tipos de comunicación entre controladores y módulos externos, en este apartado la principal necesidad ha sido conectar Arduino Uno y Arduino Yún, por lo que se han tratado tres técnicas de comunicación.

- Comunicación Serie:

Este tipo de comunicación es la utilizada para comunicar las placas Arduino al ordenador por el puerto USB. También se puede realizar una comunicación serie mediante los pines 0 y 1 en el caso de las placas Uno y Yún. Sin embargo para utilizar estos pines hay que interrumpir la comunicación por el puerto USB, lo que dificulta el trabajo de forma considerable.

La comunicación serie es una comunicación asíncrona, es decir, no existe una señal de reloj común entre los dispositivos, con lo que hay que considerar que el mensaje debe sincronizar ambos dispositivos previamente con dos bits (Start y Stop) para anunciar y concluir el mensaje.

Otro inconveniente es que no se verifica si el dato llega o no al otro dispositivo. En el caso de la aplicación que se esta trabajando en este proyecto no conviene que se pierdan datos ya que sería muy difícil resincronizar la señal.

Por estas razones se ha descartado la comunicación serie para conectar las placas Arduino.

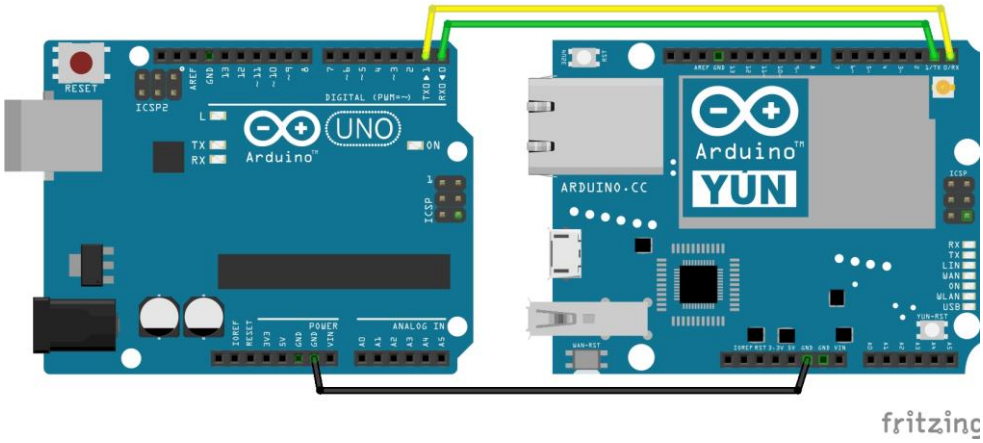


Figura 4-8. Comunicación Serie. [2]

- Comunicación SPI:

La comunicación SPI (Serial Peripheral Interface) fue desarrollada por Motorola y tuvo bastante éxito en el ámbito industrial. Se diseñó pensando en la comunicación de un microcontrolador con varios periféricos.

Es una comunicación síncrona de 4 hilos que alcanza velocidades muy rápidas y que puede enviar y recibir datos al mismo tiempo. Esto es una ventaja pensando en la aplicación de este proyecto.

El primero de los hilos es el SCK que sincroniza todos los dispositivos que intervienen en la comunicación para eliminar la incertidumbre que se tenía en la comunicación serie. Envía una señal de reloj común por la que se rigen todas comunicaciones por el bus.

Luego están las líneas MISO (Master In Slave Out) y MOSI (Master Out Slave In), por donde viajan los datos enviados y recibidos de los diferentes dispositivos.

Por último, la línea SS (Slave Select) que selecciona el periférico con el que se comunica el Master. Normalmente la selección de esclavo se realiza poniendo a nivel bajo la señal SS, que suele estar a nivel alto por defecto para bloquear la comunicación. Cada periférico conectado al bus necesita una línea SS por lo que no conviene tener muchos dispositivos comunicados entre sí utilizando este tipo de comunicación.

La razón de no haber usado este tipo de comunicación ha sido precisamente la necesidad de conectar 4 hilos entre las placas Arduino. Teniendo en cuenta que también se va a utilizar la pantalla táctil, reservar 4 pines en cada placa para crear un bus de este tipo no es posible.

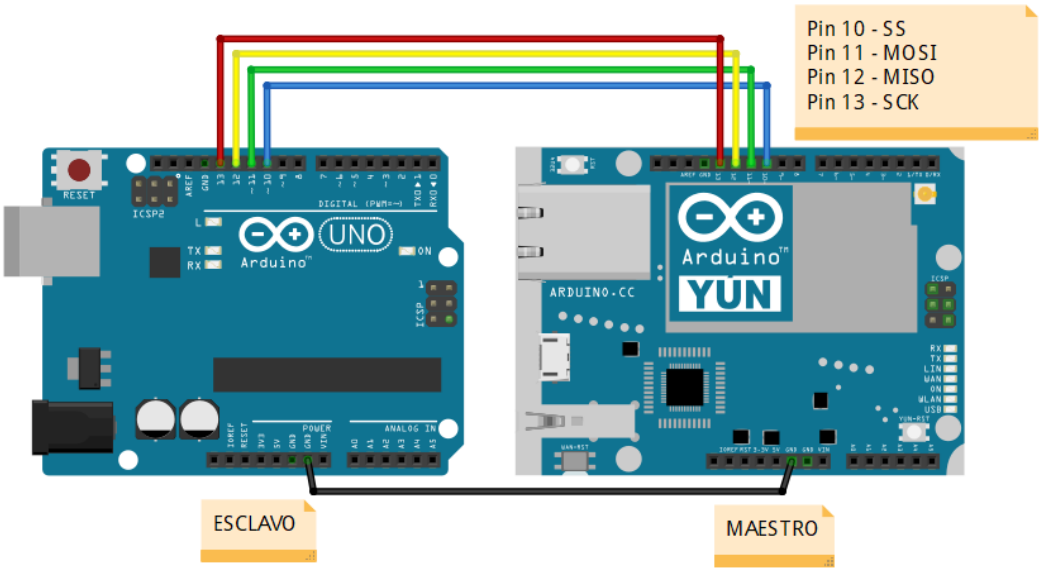


Figura 4-9. Comunicación SPI. [2]

- Comunicación I2C o TWI:

La comunicación I2C (Inter-Integrated Circuits) o TWI (Two Wires Interface) se desarrolló con el objetivo de crear un estándar con el que todos los módulos y circuitos integrados tuviesen unos valores de tensión y protocolo de comunicación comunes para no tener que trabajar con infinidad de interfaces y drivers.

Este estándar contempla, además de la alimentación Vcc y la tierra GND, dos hilos de comunicación. El SDA para el envío y recepción de datos, mientras que el SCL contiene la señal de reloj.

Cada dispositivo conectado a este bus tiene su dirección exclusiva de 7 bits, permitiendo teóricamente un total de hasta 128 dispositivos comunicados por el mismo bus.

Al menos debe haber un dispositivo máster que controla la señal de reloj, por lo que, no se requiere una sincronización estricta mientras que no hay comunicación. Este bus permite varios másters conectados pero no más de uno realizando la comunicación al mismo tiempo.

Otra de las características de este protocolo es el arbitraje y la detección de colisiones que proporciona a la comunicación. Esto asegura que se perderán bastantes menos datos que en los otros dos tipos de comunicación visto anteriormente.

La idea es que todos los componentes se conecten al bus en paralelo y que nunca haya dos dispositivos máster a la vez. El resto de módulos que no van a actuar como máster se configuran como esclavos.

Es necesario colocar dos resistencias pull-up conectadas a SDA y SCL, ya que el bus está activo a nivel bajo, sin embargo, en el caso de las placas Arduino empleadas en este proyecto, vienen dos pines incorporados precisamente con el propósito de establecer este tipo de comunicación en los que estas resistencias ya vienen integradas.

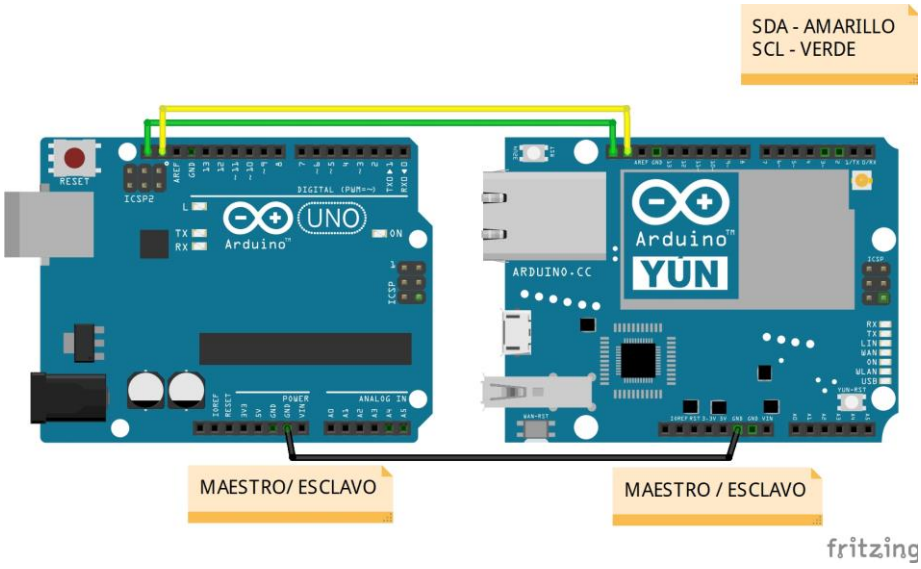


Figura 4-10. Comunicación I2C. [2]

Otra razón para haber elegido esta comunicación es la facilidad de programación que brinda la librería *Wire*. Esta librería incluye funciones y ejemplos para configurar las placas como máster y como esclavo, lo que ha facilitado mucho el trabajo a la hora de comunicar el montaje con este protocolo.

En el caso de este montaje, ambas placas, Arduino Yún y Arduino Uno, han sido conectadas como máster y como esclavo, para que ambas puedan tomar el control de la comunicación según la función que se esté realizando.

En el caso de la comunicación I2C, la velocidad de transmisión de datos es más alta que la de la comunicación serie, pero es más lenta que la comunicación SPI. Sin embargo, las razones de haber utilizado esta comunicación y no la SPI han sido el menor número de pines a ocupar, la incorporación de la librería *Wire* de Arduino y que la velocidad de transmisión es admisible en este proceso.

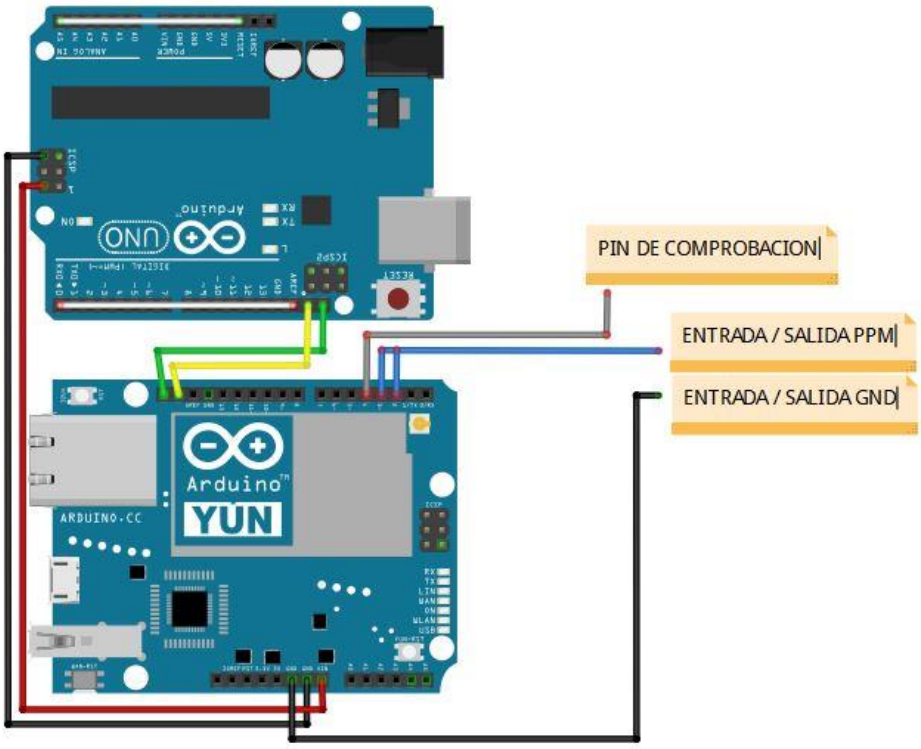


Figura 4-11. Diagrama de conexión I2C en Arduino Yún + Arduino Uno. [2]

4.3.2 Programación de la interfaz gráfica

La pantalla táctil ensamblada en el montaje de este prototipo ha sido la forma con la que se pretende dar a este proyecto un manejo autónomo y atractivo al usuario. Han sido varias las razones que han motivado el uso de la pantalla táctil en este proyecto.

- Este dispositivo se ha incluido en el desarrollo de este proyecto como medio de comunicación entre el usuario y el dispositivo grabador-reproductor. Se busca así la facilidad de manejo por cualquier persona, usando funciones de control sencillas y tratando de no perder de vista la aplicación final para la que se ha pensado este dispositivo que es la grabación y reproducción de señales en el control de robots animatrónicos.
- Darle un aspecto de producto final al prototipo, y que pueda ser usado de forma más o menos intuitiva sin necesidad de conocimientos específicos acerca de las señales tratadas.
- Aprender a trabajar con este tipo de plataformas y realizar un primer acercamiento a la tecnología táctil que proporciona al dispositivo un valor añadido al disponer de una interfaz flexible y atractiva para el usuario.

Como ya se mencionó anteriormente al describir este dispositivo en el capítulo de selección de hardware, esta pantalla en concreto, trabaja de la misma forma que lo hacen los dispositivos táctiles desarrollados por la empresa Adafruit, con lo que funciona con las mismas librerías proporcionadas por este fabricante. Aunque existen muchas más, en este proyecto se han usado las siguientes:

- <Adafruit_GFX.h>

  Esta librería proporciona la sintaxis necesaria en Arduino para trabajar con la generación de gráficos e imágenes en la pantalla. Las funciones de esta librería permiten al desarrollador dibujar formas geométricas, colorear espacios, animar imágenes y cargar fotografías entre otras funciones.

  Contiene funciones para definir la orientación de la pantalla, refrescar la imagen y para trabajar con variables de control que gestionan tanto colores (definidos en formato RGB) como movimientos y aparición y desaparición de formas.

- <Adafruit_TFTLCD.h>

  Es la librería que trabaja con la gestión de pines del dispositivo para adaptar la capa física de la pantalla a la librería GFX anterior, común para todos los dispositivos.

  Define parámetros como el tamaño de la pantalla, la gestión de la comunicación con la placa Arduino y también gestiona el slot micro SD que tiene incorporado este módulo.

  La diferencia con la librería gráfica es que esta librería está enfocada a la gestión interna del dispositivo, más que a la parte visual y cualitativa del mismo.

- <TouchScreen.h>

  En este caso, esta librería gestiona las opciones táctiles de la pantalla. Mide parámetros como la resistencia generada al tocar la pantalla y calcula la posición del punto donde se ha realizado la pulsación o el desplazamiento.

  Da la opción de comprobar la fuerza o el tiempo de la pulsación y proporciona funciones para determinar la orientación del panel táctil. Esta opción es importante tenerla en cuenta, ya que será mucho más fácil trabajar si la parte gráfica y la parte táctil tienen sus ejes X e Y orientados de la misma forma.

Con el manejo de estas librerías y la aplicación de criterios lógicos con arduino se ha dibujado una interfaz de usuario que recibe las órdenes según la posición del punto donde se identifique la pulsación.

Antes de pasar a la programación se han hecho diferentes diseños de interfaz de usuario que se muestran a continuación. Se han tenido en cuenta funciones que luego no se han llegado a implementar, por lo que se han ido eliminando o añadiendo elementos para que la interfaz sea sencilla y operativa.

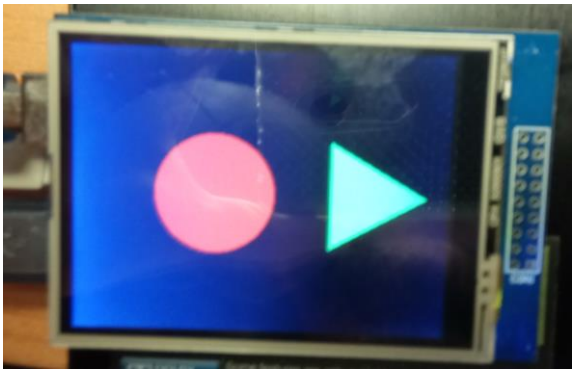


Figura 4-12. Pruebas de diseño de la interfaz gráfica 1

Figura 4-13. Pruebas de diseño de la interfaz gráfica 2

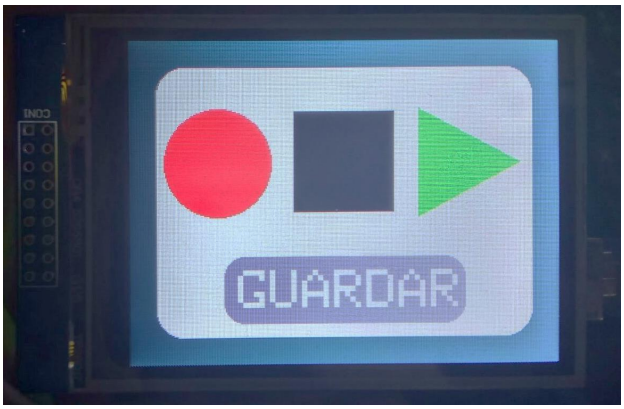


Figura 4-14. Pruebas de diseño de la interfaz gráfica 3

Figura 4-15. Pruebas de diseño de la interfaz gráfica 4

Para la interfaz de usuario final se ha utilizado código que incluye los distintos procesos como funciones, arbitrados según el valor entregado por una variable global. Una vez se han conseguido las imágenes y las variables necesarias para el control del dispositivo se han incluido las funciones de la pantalla en el código final de la placa Arduino Uno.

```
//RECTANGULO
tft.fillRoundRect(punto X0, punto Y0, longitud, altura, color);

//RECTANGULO CON ESQUINAS REDONDEADAS
tft.fillRoundRect(punto X0, punto Y0, longitud, altura, radio, color);

//CIRCULO
//tft.fillCircle(centro X, centro Y, radio, color);

//TRIANGULO
//tft.fillTriangle(punto 1X, punto 1Y, punto 2X, punto 2Y, punto 3X, punto 3Y, color);
```

```
//INSERTAR TEXTO EN PANTALLA
tft.setTextSize(3);              //DEFINIR TAMAÑO
tft.setTextColor(WHITE);         //DEFINIR COLOR
tft.setCursor(45, 190);          //POSICIONAR CURSOR PARA EMPEZAR A ESCRIBIR
tft.println("GRABADORA PPM");    //MENSAJE A IMPRIMIR
```

Figura 4-16. Interfaz gráfica

4.3.3 Eliminación del Canal 0 y compresión de datos a 1 byte

Como ya se ha visto en otros apartados, la velocidad de transmisión de los datos debe ser optimizada al máximo, ya que cualquier retraso en alguno de los procesos puede desvirtuar la medición de los pulsos e introducir errores en la grabación.

Esta optimización pasa por una sincronización muy fina de los elementos que intervienen en este montaje, sin embargo, otro de los puntos donde se puede realizar esta optimización es en los datos y en reducir al máximo el peso de los mismos, para agilizar los procesos y evitar tener que recomponer datos cada vez que se realiza una comunicación.

En este proyecto se trabaja con diferentes tipos de datos de Arduino. Los más básicos son:

- Caracteres (16 bits): ***char***
- Enteros (16 bits): ***int***

También se trabaja con arrays, matrices, cadenas y enteros sin signo, entre otros. Estos tipos de datos se han usado según conveniencia del proceso a aplicar.

- Lectura y escritura en SD: *char* y *String*.
- Valores temporales de la señal: *int, unsigned int, arrays[i], matrices[i][j]*.

Esto implica que se hayan tenido que usar funciones que Arduino incorpora para la gestión y transformación de unos datos en otros.

Sin embargo, no se ha tenido en cuenta hasta ahora el peso de estos datos. Esto se debe a que la comunicación entre dispositivos envía y recibe datos byte a byte. Esto implica que un dato tipo entero debe ser enviado en dos veces y recomponer el dato en el dispositivo receptor, lo que duplica el trabajo en cada dispositivo. Por

esta razón se ha decidido tratar de comprimir los datos para que solo sea necesario 1 byte para definirlos.

Recordando los datos de las tramas grabadas:

```
12516/0
1464/1
1500/2
1008/3
1512/4
1000/5
1000/6
```

Figura 4-17. Trama almacenada

Los datos obtenidos de la señal PPM han sido tomados en microsegundos. Volviendo a plantear las características de la señal:

- Pulsos en cada canal variables entre 1 ms y 2 ms.
- Pulso vacío de sincronización caracterizado por el Canal 0.
- Periodo de la señal de 20 ms.

Con el objetivo de reducir al mínimo los datos obtenidos se plantea como primera opción **eliminar el Canal 0**. Es el dato más grande y se puede calcular fácilmente:

$$Canal\ 0 = Periodo - \sum_{n=1}^{n=6} canales\ n$$

$$Canal\ 0 = 20000\ - (1464 + 1500 + 1008 + 1512 + 1000 + 1000)$$

$$\boldsymbol{Canal\ 0 = 12516\ ms}$$

Con esta sencilla fórmula se puede eliminar el dato del Canal 0 y trabajar solo con los pulsos que contienen los datos de cada canal.

Seguidamente para trabajar con el resto de datos se tiene en cuenta que, para almacenarlos en 1 byte, los datos no pueden salirse del rango de entre 0 y 255, teniendo en cuenta que estos datos siempre serán positivos.

De esta manera se deduce que teniendo una diferencia de 1000 microsegundos entre 1 ms y 2 ms se puede dividir esta diferencia entre 4 para así obtener valores de 0 a 250.

$$Dato\ (1\ byte) = \frac{Dato\ en\ \mu s - 1000}{4} = valor\ entre\ 0\ y\ 250$$

$$Canal\ 1\ (1\ byte) = \frac{1464 - 1000}{4} = \mathbf{116}$$

Evidentemente esta operación de división sacrifica la exactitud de la medición temporal en unos pocos microsegundos, sin embargo, este tipo de señal tiene variaciones durante la transmisión que rondan esa magnitud.

De esta forma, a la hora de transmitir datos, se consigue que por cada byte enviado se reciba un dato completo en el otro dispositivo. Este dato deberá ser transformado de nuevo para seguir trabajando con él de cara a la generación de la nueva señal.

Con estos cambios en el tratamiento de datos se han obtenido los siguientes resultados reales en las pruebas físicas:

1. La grabación de la señal PPM pasa a muestrear 1 de cada 3 tramas, y no 1 de cada 4 como se muestreaban anteriormente.
2. La comunicación entre dispositivos es más fluida y rápida.

4.3.4 Sincronización de las órdenes de la pantalla

Al haber incluido la pantalla táctil en el montaje, se ha hecho necesario sincronizar las órdenes entre ambos dispositivos.

La pantalla se ha ensamblado en los pines del Arduino Uno, y mediante los pines SDA, SCL y GND se ha habilitado un bus I2C entre ambos. Arduino Yún ha sido el encargado de realizar la función de grabación y reproducción.

Las diferentes figuras en la pantalla corresponden, cada una, a una orden al dispositivo. Al recibir una pulsación, en la zona del panel táctil delimitada para dicha pulsación, se recibe un valor diferente en una variable habilitada para ello. De esta forma las órdenes enviadas por el panel táctil se han identificado con la variable *a* cuyo valor se ha definido de la siguiente forma.

- RECORD cuando *a=1*. Pulsando un círculo rojo.
- STOP cuando *a=2*. Pulsando un cuadrado negro.
- PLAY cuando *a=3*. Pulsando un triángulo verde.
- ERROR cuando *a=0*. Pulsando en cualquier otra zona de la pantalla.

Figura 4-18. Órdenes de la interfaz gráfica

En todos los casos el proceso a realizar por el dispositivo es el siguiente:

1. Se le da valor a la variable *a*.
2. Se imprime en la pantalla la orden pulsada.
3. Se envía el valor de *a* por el bus I2C al Arduino Yún, encargado de ejecutar la orden.
4. Se refresca la pantalla para eliminar la orden impresa.
5. Se envía al monitor serie del ordenador la función que se está ejecutando.

6. Comienza la ejecución de la función deseada.

4.3.5 Conclusiones

Este nuevo montaje incluye las mismas funciones programadas con anterioridad en el Arduino Yún, pero se ha trabajado con la comunicación I2C y la interacción del usuario con la pantalla táctil. Las limitaciones temporales en materia de almacenamiento en la SD siguen impidiendo una optimización de la reproducción, sin embargo se ha conseguido reducir el número de tramas perdidas en el muestreo de la señal gracias a la compresión de los datos a tipo byte.

También se ha conseguido integrar un dispositivo con una interfaz táctil capaz de grabar y reproducir señales de control. No obstante sigue existiendo el inconveniente de la limitación de tiempo en el modo reproducción.

4.4 Implementación en Arduino Uno + Arduino Uno

En esta implementación se ha querido trabajar con el slot SD incorporado en el módulo de la pantalla táctil. Arduino Yún fue seleccionado en un principio para trabajar con su slot SD incorporado y para explorar las opciones IoT que el dispositivo ofrece, sin embargo al avanzar en el desarrollo se ha observado que sus características están enfocadas a otro tipo de aplicaciones que requieren de un sistema operativo y una conectividad potentes, no aportando mucho más que la placa Arduino Uno a esta aplicación.

Pensando también en el requisito de hacer un dispositivo de bajo coste se ha sustituido el Arduino Yún por otra placa Arduino Uno para trabajar en paralelo con dos placas de la misma forma que se ha hecho en el apartado anterior.

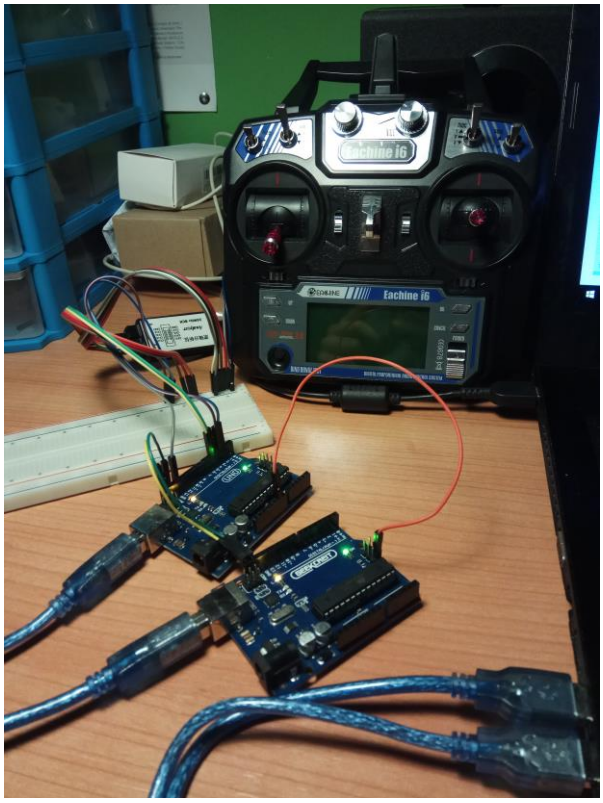


Figura 4-19. Implementación del sistema en Arduino Uno + Arduino Uno.

4.4.1 Slot SD integrado en la pantalla

Al haber pasado de Arduino Yún a Arduino Uno ha sido necesario modificar los programas descritos anteriormente, ya que Arduino Uno gestiona la lectura y escritura de datos con la librería <*SD.h*>. Esta librería tiene casi las mismas funciones que la librería *Bridge* usada para Yún, sin embargo, los nombres de las mismas y algunos de los parámetros son diferentes por lo que hay que considerar ciertos cambios en el script.

Al tratarse de un módulo externo a la placa es necesario incluir la librería <*SPI.h*> y definir el pin de selección (Slave Selection o Chip Selection). En este caso se ha usado el pin 10, que es el que esta indicado en el reverso del módulo táctil.

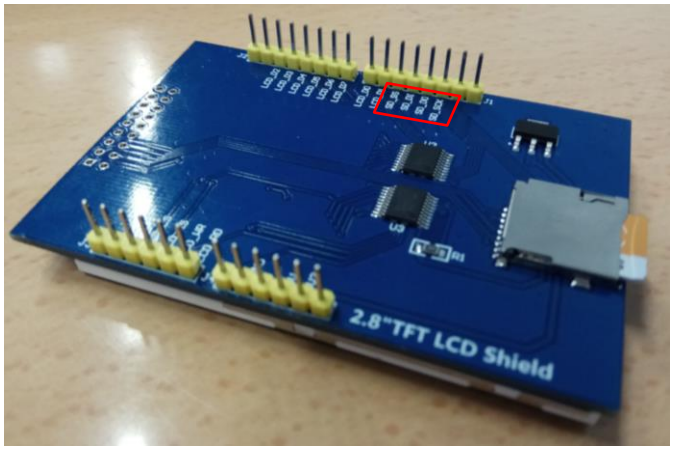


Figura 4-20. Reverso del módulo táctil. Pines del slot SD.

Tras hacer pruebas con este slot se han obtenido peores datos temporales en la lectura y escritura de la tarjeta SD. Aun optimizando los datos para que cada canal ocupe 1 byte en cada trama, este tipo de memoria no es capaz de asumir un periodo tan corto de lectura y escritura. Así pues, se puede pensar que la opción con Arduino Yún es mejor en este aspecto.

4.4.2 Conclusiones

Esta opción es, de todas las contempladas, la que más se acerca a la arquitectura compacta pensada en un principio y la que mejor cumple el requisito del bajo coste, sin embargo tras probar de nuevo el almacenamiento en tarjetas SD se vuelve a confirmar que este tipo de almacenamiento no es el adecuado para esta aplicación.

Uno de los principales retos que se plantea a partir de este punto es tratar de transmitir la información a otro equipo con una memoria interna de acceso rápido y capacidad suficiente para gestionar los datos a la velocidad que requiere la aplicación.

5 SOLUCIÓN FINAL

Tras verificar las limitaciones temporales que la tarjeta SD transmite a esta aplicación se han barajado otras opciones de almacenamiento como la utilización de memorias USB que trabajan de forma muy parecida, o tratar de almacenar los datos en un pequeño buffer en la memoria interna de las placas Arduino, ambas sin éxito. En este capítulo se expone la solución que se le ha dado al problema del almacenamiento de los datos en memoria.

5.1 Lectura y envío de datos por el puerto serie al ordenador

Durante todo el desarrollo se ha estado trabajando con el puerto serie para cargar nuevos programas y trabajar con el monitor serie incorporado en el IDE de Arduino que facilita la visualización de cadenas de caracteres por pantalla de forma sencilla.

Por tanto, una vez se ha contemplado la opción de usar un equipo externo para la recopilación de datos, se plantea la posibilidad de extraer esta información mostrada por pantalla directamente y de forma automática a un archivo de texto.

5.1.1 Software libre. Programa CoolTerm

Existen bastantes alternativas. En el caso de este proyecto se ha optado por un programa de software libre ya desarrollado llamado *CoolTerm* que precisamente realiza esta tarea.

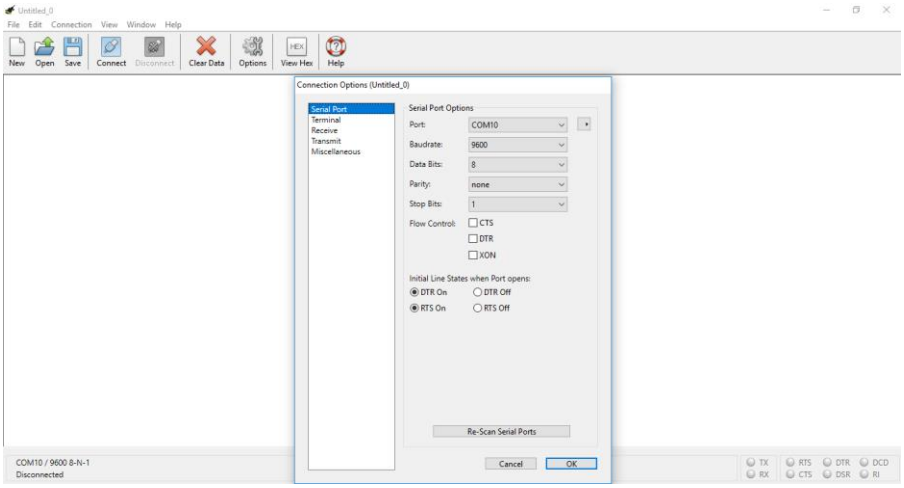


Figura 5-1. Software CoolTerm: Conexión a Arduino por puerto serie .

Conectando las placas Arduino, especificando el puerto serie y sincronizando el baud rate de transmisión de datos, en pocos minutos, se consigue visualizar los datos entregados al monitor serie a través de este programa.

Seguidamente se prepara el archivo donde se irán capturando los datos, y este software carga las cadenas de caracteres en un archivo especificado *.txt* mientras que se muestran por pantalla.

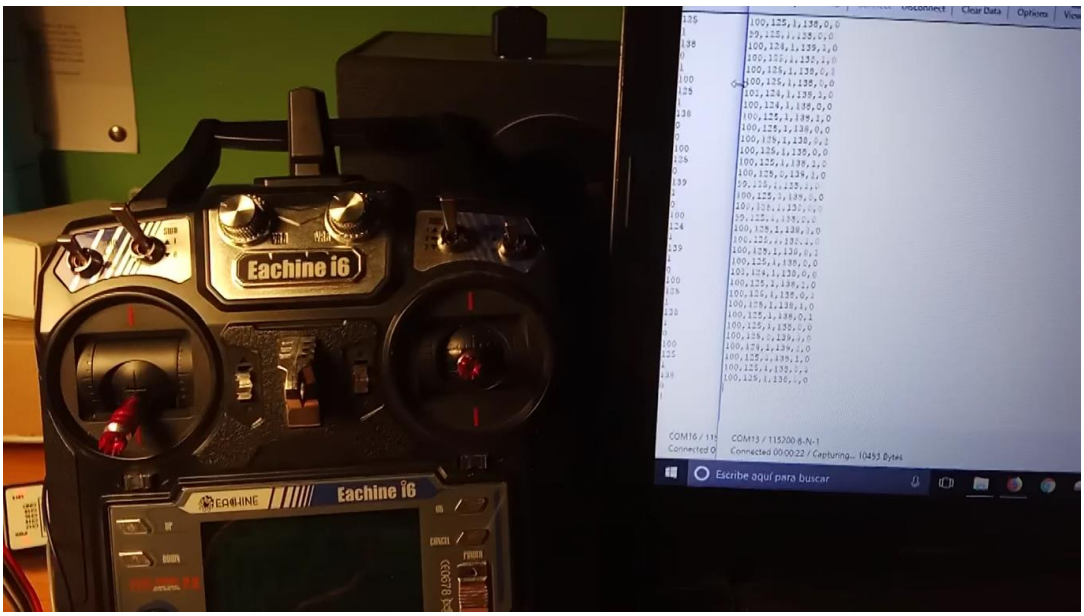


Figura 5-2. Software CoolTerm: Captura de datos por puerto serie .

5.1.2 Otras opciones

Hasta llegar a este programa se han encontrado muchas otras opciones como mostrar en tiempo real gráficas en Excel usando VisualBasic. También se desarrollan aplicaciones propias en Java, y en otros lenguajes, mucho más específicas para cada caso y aplicación concretos.

Es interesante seguir barajando estas opciones o incluso desarrollar un software propio de la aplicación como futura líneas de trabajo.

5.2 Tratamiento, lectura y edición de datos

Los datos obtenidos se han ido visualizando y conociendo durante el desarrollo. Trabajando en diferentes casos se ha podido ver el comportamiento de los mismos y las gráficas que se han ido observando con el analizador lógico han clarificado mucho el funcionamiento de este tipo de señal de control.

No obstante, es necesario ver si tras el proceso de grabación y manipulación de la señal, se pueden obtener datos interpretables y más visuales de la señal que se correspondan con el control realizado durante la grabación, y que puedan ser utilizados para la reproducción.

Para ello se ha trabajado en Excel con los datos graficando cada canal de forma independiente y obteniendo los siguientes patrones de control en los diferentes canales.

Estas imágenes corresponden a una grabación concreta en la que se han utilizado todas las palancas e interruptores al mismo tiempo para observar la evolución de los datos de cada canal durante un manejo real de la emisora.

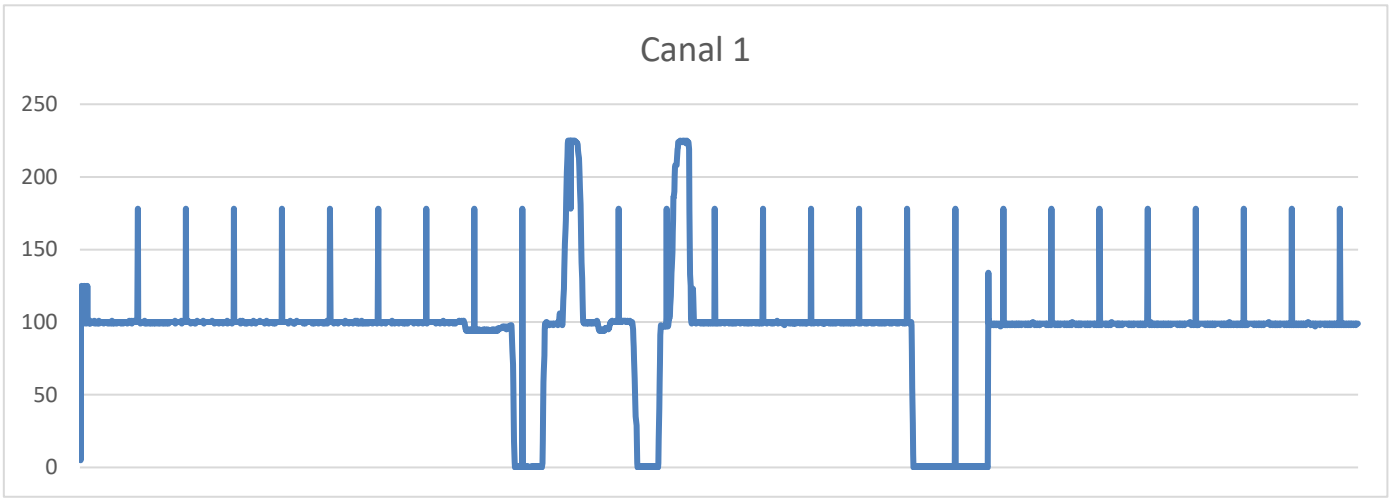


Figura 5-3. Analisis en Excel: Canal 1.

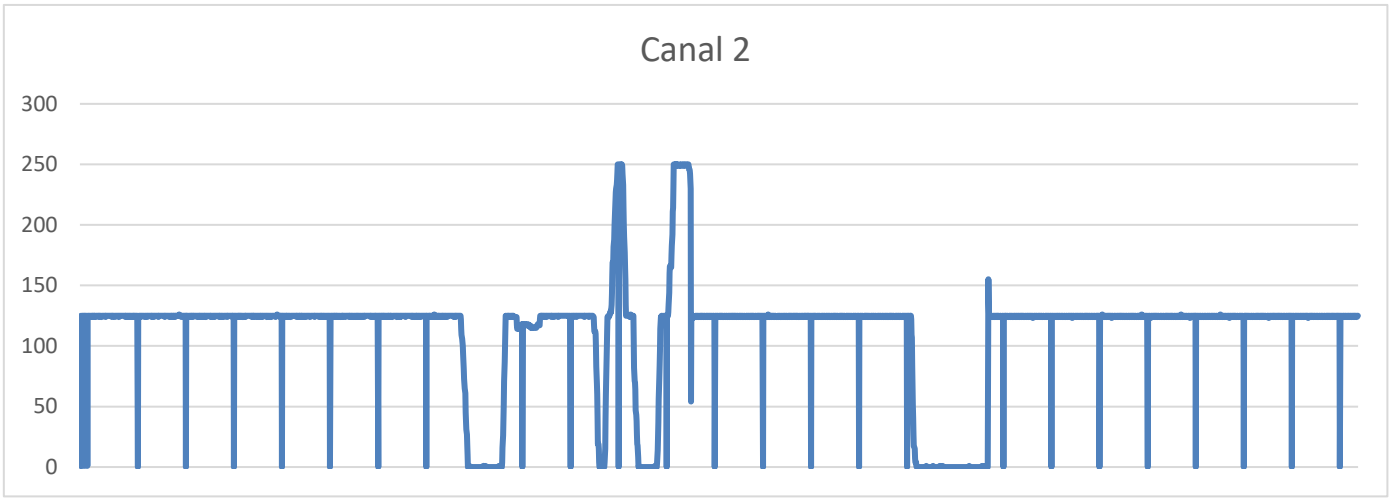


Figura 5-4. Analisis en Excel: Canal 2.

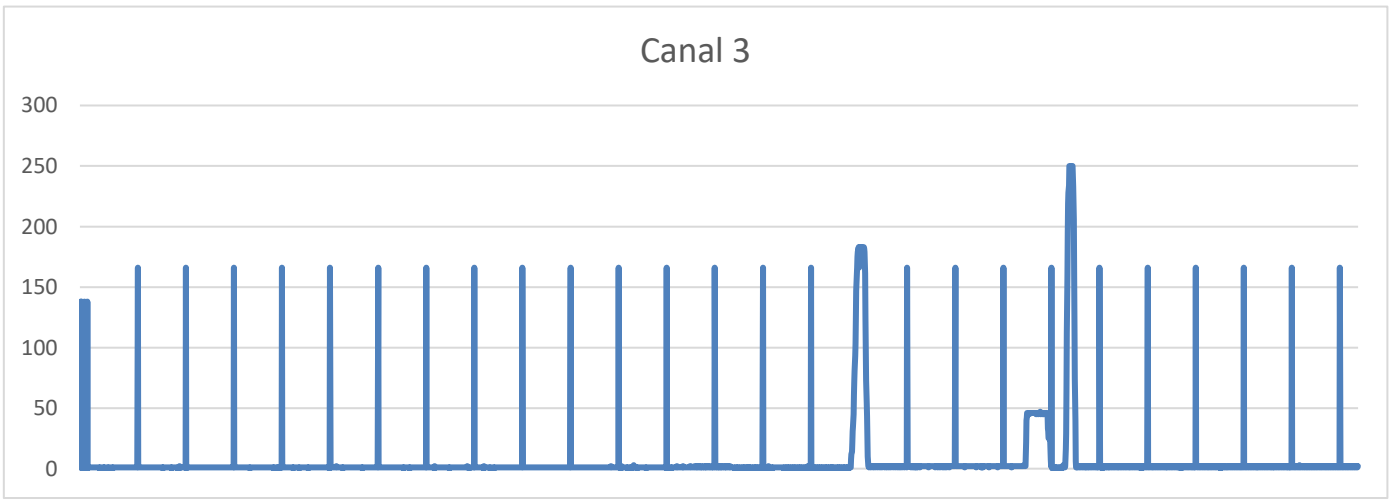


Figura 5-5. Analisis en Excel: Canal 3.

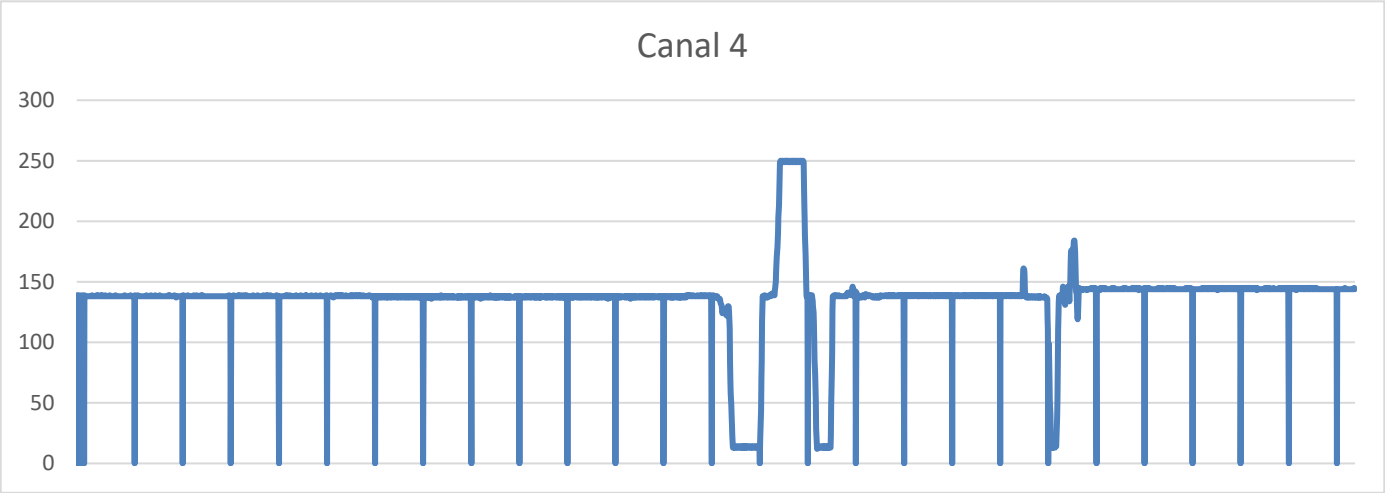


Figura 5-6. Analisis en Excel: Canal 4.

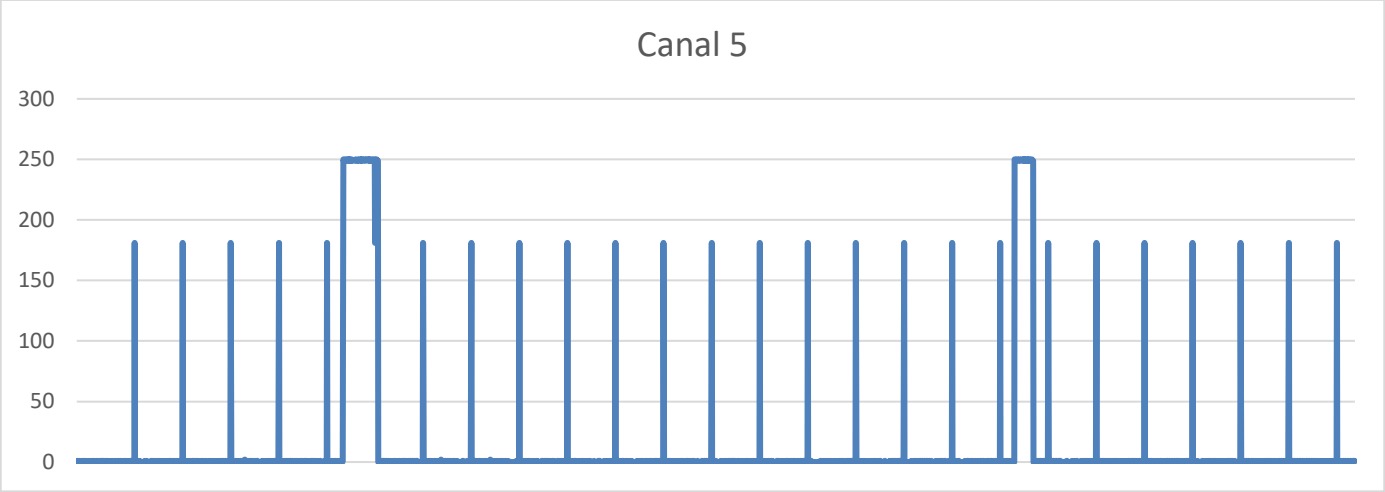


Figura 5-7. Analisis en Excel: Canal 5.

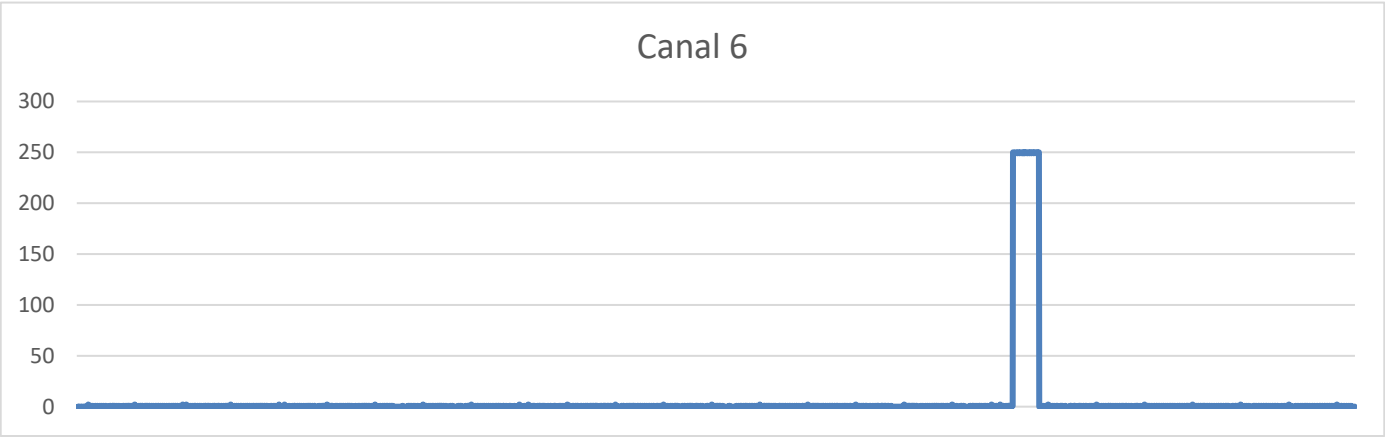


Figura 5-8. Analisis en Excel: Canal 6.

Destacar que los valores del eje vertical son directamente los datos comprimidos, y por tanto adimensionales. No se corresponden con el valor de la duración de los pulsos, sin embargo con una simple transformación se obtendría la misma gráfica en el orden de magnitud correcto.

Aunque se puede observar un patrón más o menos lógico, se observa un glitch periódico. A nivel de estudio de la señal se han suprimido estos glitches para observar que las señales obtenidas tienen una forma interpretable que da una idea del control que se hizo durante la grabación.

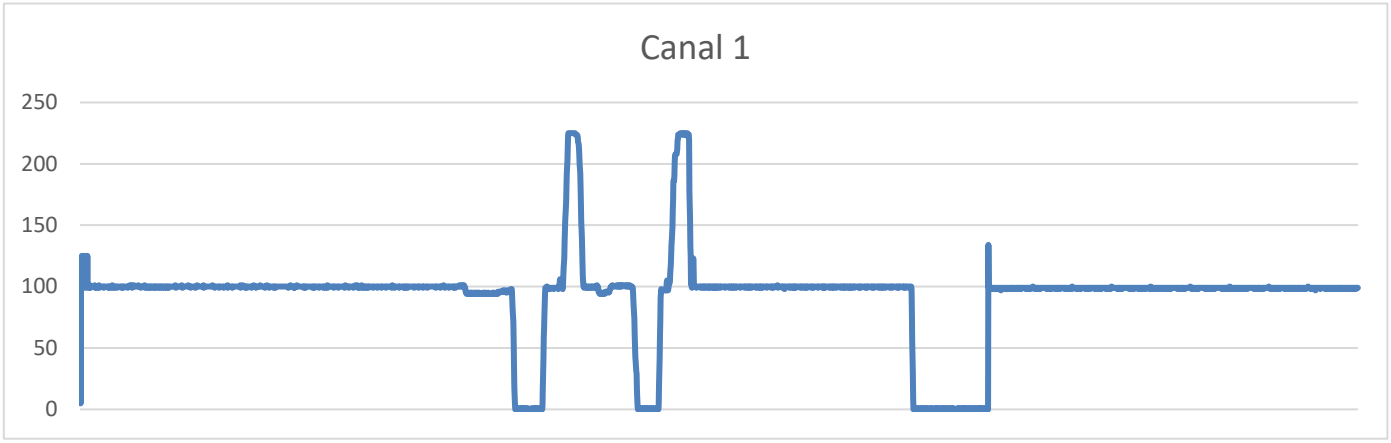


Figura 5-9. Analisis en Excel: Canal 1 filtrado.

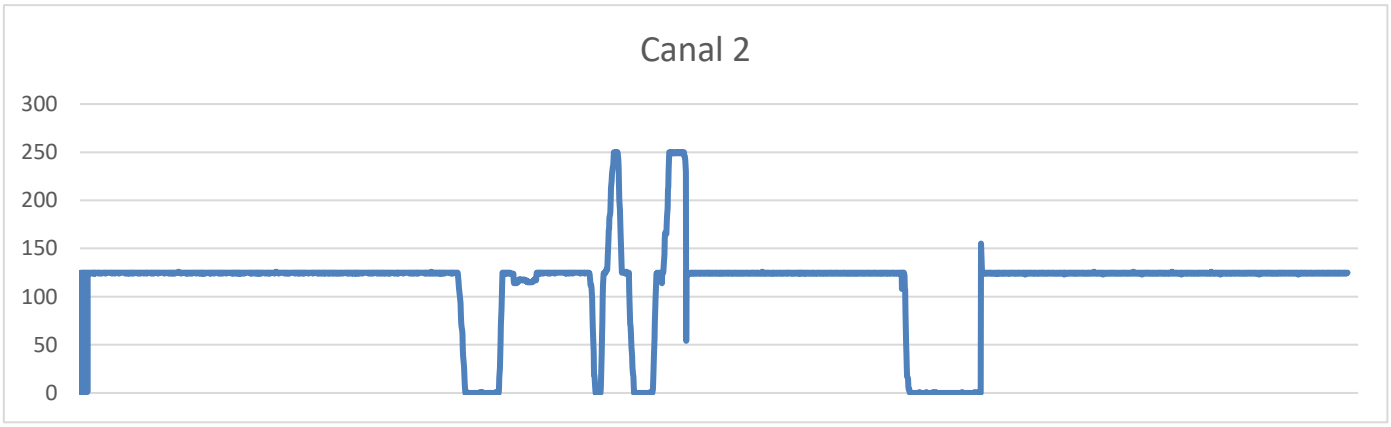


Figura 5-10. Analisis en Excel: Canal 2 filtrado.

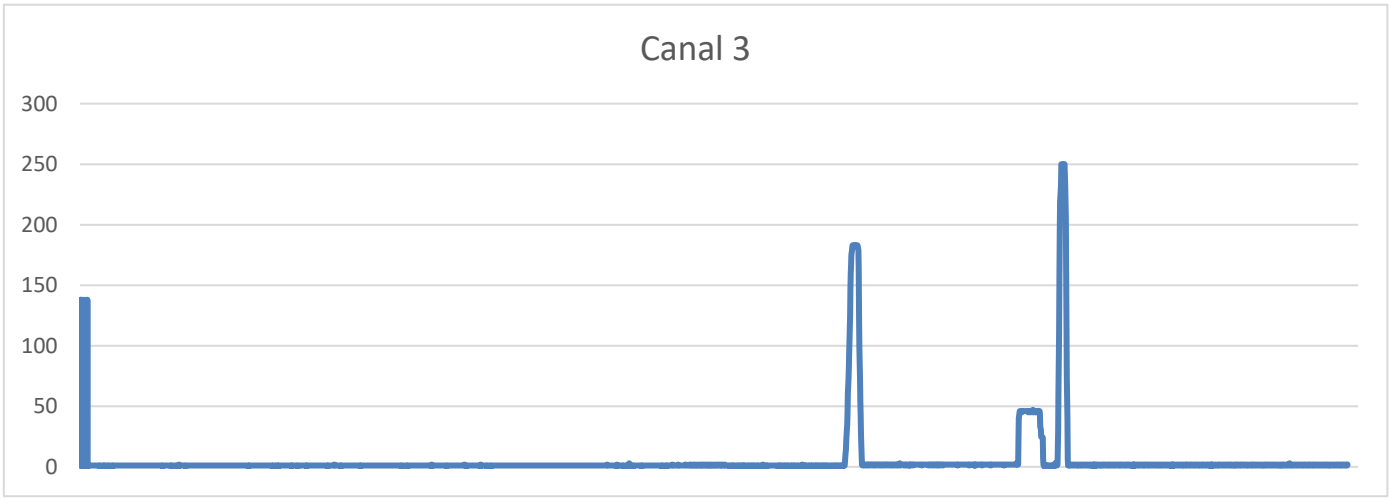


Figura 5-11. Analisis en Excel: Canal 3 filtrado.

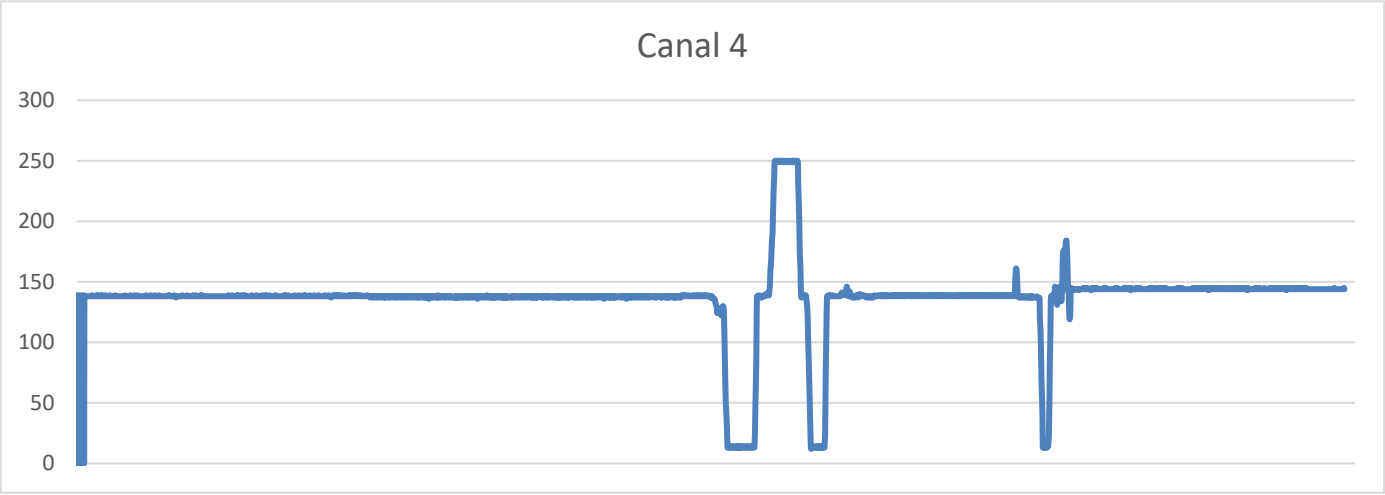


Figura 5-12. Analisis en Excel: Canal 4 filtrado.

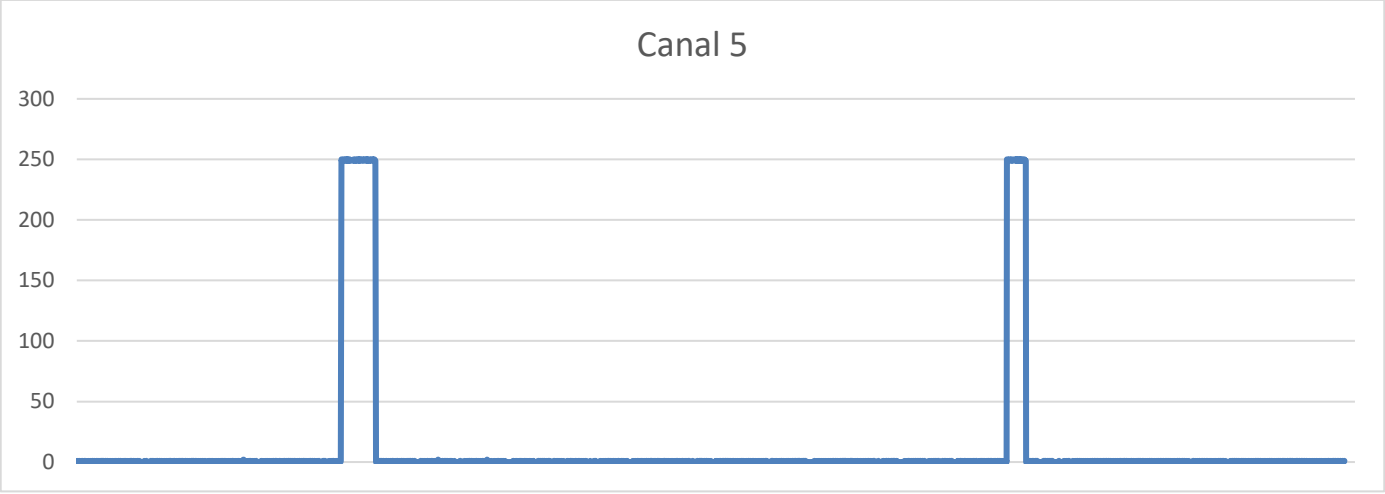


Figura 5-13. Analisis en Excel: Canal 5 filtrado.

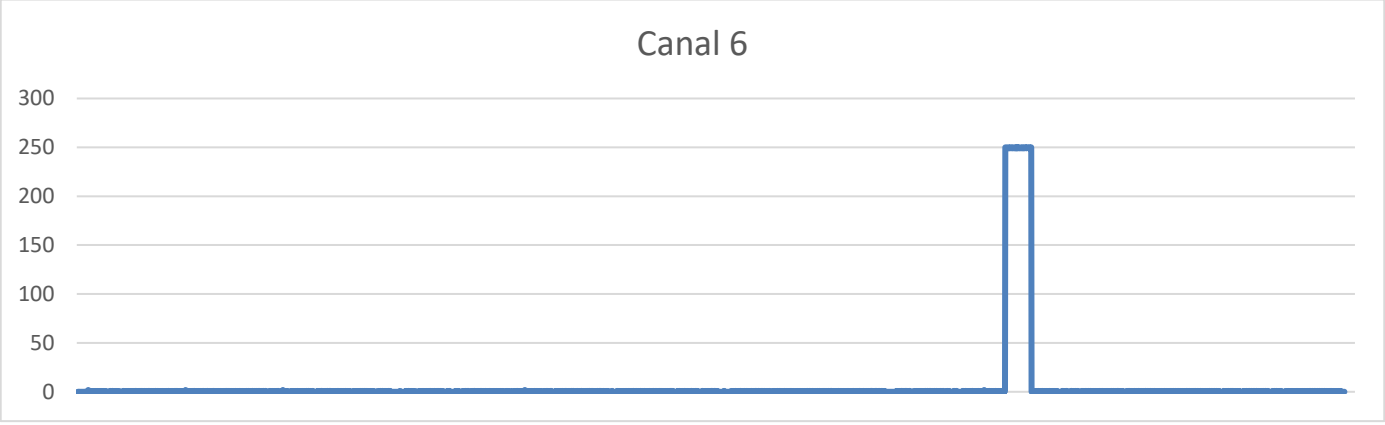


Figura 5-14. Analisis en Excel: Canal 6 filtrado.

Estos saltos en la señal pueden ser eliminados mediante un software de filtrado. Este estudio también da una idea de la posibilidad de desarrollar un software para edición de patrones grabados de forma parecida al trabajo que se realiza con señales de audio, bien modificando los datos, o bien de forma gráfica.

Gráficamente también se puede observar cómo el movimiento de los canales se corresponde con el movimiento de control en las palancas de la emisora. En algunas ocasiones el movimiento se ha realizado en cada eje, pero en otros la posición de la palanca se ha variado en diagonal, por eso las gráficas varian al mismo tiempo.

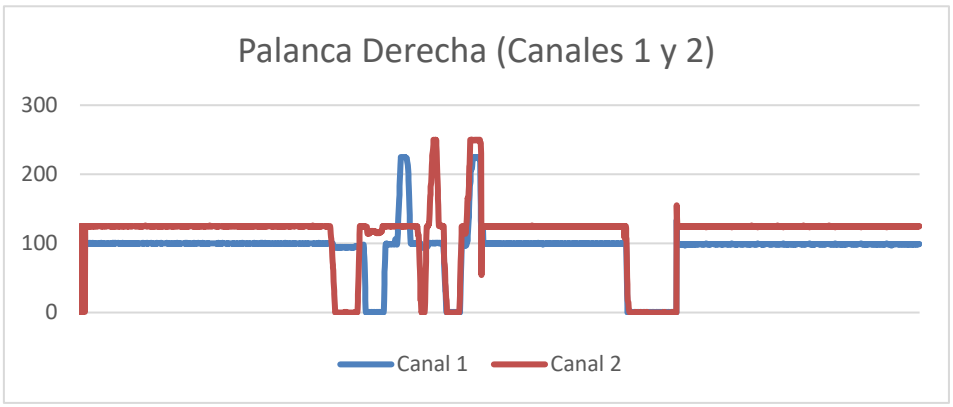
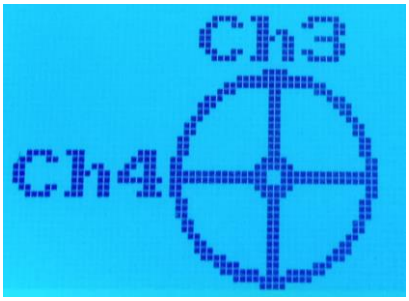


Figura 5-15. Analisis en Excel de palanca derecha: Canales 1 y 2.



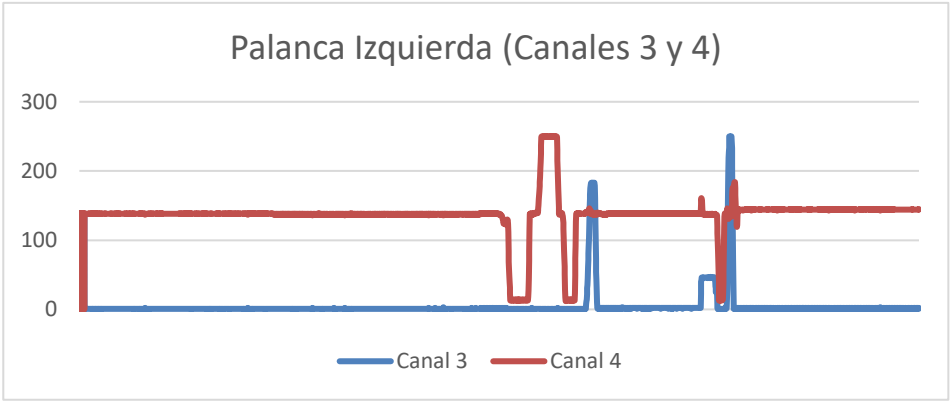


Figura 5-16. Analisis en Excel de palanca izquierda: Canal 3 y 4 filtrado.

En el caso de los canales 5 y 6 se observa como, al ser interruptores solo muestran nivel bajo o nivel alto.

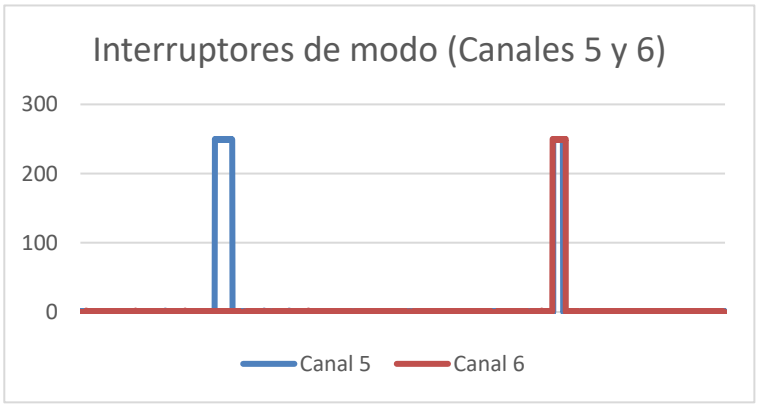


Figura 5-17. Analisis en Excel de interruptores de modo: Canales 5 y 6 filtrado.

5.3 Reproducción de señales PPM mediante interrupciones

Tratando de mejorar el modo reproducción del dispositivo se plantea la opción de sustituir los *delays* por interrupciones temporales que cada tiempo prefijado cambie el estado de la señal sin necesidad de estar parando la ejecución del programa constantemente.

```
void flanco()
{
  if (nivel == HIGH) {
    nivel = LOW;
    canal++;
    }     else {
    nivel = HIGH;
  }
  digitalWrite(4, nivel);
```

Cada dato es leido por una función que establece el valor necesario en el *Timer1* de Arduino. Este temporizador, directamente ligado al reloj de la placa, permite obtener la duración necesaria del canal.

```
  if (canal > ncanales){
    canal = 0;
  }
if (nivel == LOW) {
    Timer1.setPeriod(400);
    }     else {
    Timer1.setPeriod(ppm[canal]);
  }
```

La función *Timer1.setPeriod()* lee la duración del pulso dada por el dato en cuestión. Antes de introducir el dato hay que restar la duración del gap de separación de 400 µs que se genera a nivel bajo mientras se carga un nuevo valor para el siguiente pulso.

Recordando que durante la fase de compresión de los datos se había eliminado el Canal 0, durante la carga de la trama en el vector de entrada se ha tenido en cuenta este factor, que se calcula restando el sumatorio de los pulsos de la trama al periodo de la señal. Para reproducir una trama tipo, la señal leida por el dispositivo es la siguiente:

```
100,125,0,139,1,0
```

Figura 5-18. Analisis en Excel de interruptores de modo: Canales 5 y 6 filtrado.

Para cargar las tramas es necesario un pequeño buffer que reciba las tramas del archivo de texto. En este caso las tramas son leidas del puerto serie por lo que el tiempo de carga del buffer no supone un problema.

5.4 Descomposición de la señal PPM en PWM

Para multitud de dispositivos radiocontrol existen receptores específicamente diseñados para obtener una señal PPM y descomponerla en varias señales PWM cada una por un pin al que van conectados los actuadores.

Para realizar pequeñas pruebas de concepto no se disponía de este tipo de receptores con lo que se ha implementado un código para Arduino en el que se descompone la señal PPM de entrada por un pin a varios pines de salida con señales PWM cada una de un canal.

```
int contador = 0;


void setup(){
   Serial.begin(9600);
   pinMode(7,INPUT_PULLUP);

   pinMode(0,OUTPUT);
   pinMode(1,OUTPUT);
   pinMode(3,OUTPUT);
   pinMode(5,OUTPUT);
   pinMode(6,OUTPUT);
   pinMode(8,OUTPUT);
   pinMode(4,OUTPUT);

   attachInterrupt( 4, bajada, FALLING);
   }

void loop(){
int n=0;
          if (pulseIn(7,HIGH) < 3000){            //SINCRONIZAR
            n=contador;
          }    else{
            n=1;
            contador=1;
          }
```

```
        switch(n){                    //ACTIVACIÓN Y DESACTIVACION DE PINES
          case 1:
            digitalWrite(0, HIGH);
            digitalWrite(1, LOW);
            digitalWrite(3, LOW);
            digitalWrite(5, LOW);
            digitalWrite(6, LOW);
            digitalWrite(8, LOW);
          break;

          case 2:
            digitalWrite(0, LOW);
            digitalWrite(1, HIGH);
            digitalWrite(3, LOW);
            digitalWrite(5, LOW);
            digitalWrite(6, LOW);
            digitalWrite(8, LOW);
          break;

          case 3:
            digitalWrite(0, LOW);
            digitalWrite(1, LOW);
            digitalWrite(3, HIGH);
            digitalWrite(5, LOW);
            digitalWrite(6, LOW);
            digitalWrite(8, LOW);
          break;

          case 4:
            digitalWrite(0, LOW);
            digitalWrite(1, LOW);
            digitalWrite(3, LOW);
            digitalWrite(5, HIGH);
            digitalWrite(6, LOW);
            digitalWrite(8, LOW);
          break;
```

```
          case 5:
            digitalWrite(0, LOW);
            digitalWrite(1, LOW);
            digitalWrite(3, LOW);
            digitalWrite(5, LOW);
            digitalWrite(6, HIGH);
            digitalWrite(8, LOW);
          break;

          case 6:
            digitalWrite(0, LOW);
            digitalWrite(1, LOW);
            digitalWrite(3, LOW);
            digitalWrite(5, LOW);
            digitalWrite(6, LOW);
            digitalWrite(8, HIGH);
          break;

          default:
            digitalWrite(0, LOW);
            digitalWrite(1, LOW);
            digitalWrite(3, LOW);
            digitalWrite(5, LOW);
            digitalWrite(6, LOW);
            digitalWrite(8, LOW);
          break;
        }
   }

void bajada(){                        //Interrupción cada flanco de bajada
    contador++;
   }
```

5.5 Arquitectura y dispositivo final

Figura 5-19. Montaje final.

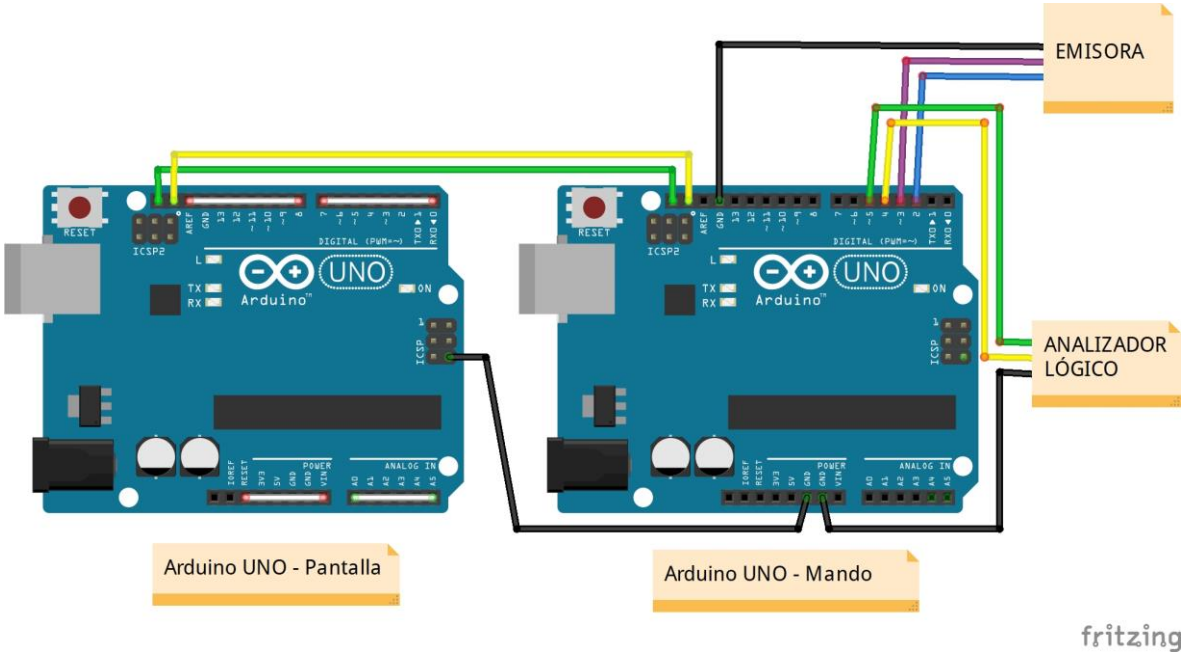


Figura 5-20. Diagrama de conexiones del montaje final. [2]

En este dispositivo final se combina todo lo aprendido durante el desarrollo para tratar de dar la mejor solución al problema planteado.

Se dispone de:

- 2 Placas Arduino Uno conectadas mediante un bus I2C y un puerto GND
- Emisora con salida PPM conectada a los pines 2 y 3 y masa conectada a uno de los puertos GND de la primera de las placas Arduino Uno.
- Pantalla táctil ensamblada a la segunda placa Arduino.
- Ordenador conectado mediante el puerto serie a la segunda placa Arduino.

- Software *CoolTerm* Sincronizado con el puerto serie en el que se conecta el dispositivo.

Para trabajar con el dispositivo se iniciará un terminal de *CoolTerm* con una captura de datos antes de empezar a grabar la señal.

El panel táctil se enciende al alimentar el dispositivo por usb y carga en la pantalla la interfaz de usuario. El panel táctil enviará las ordenes dadas mediante pulsación de las figuras a la placa Arduino Uno a la que esta ensamblada, y esta distribuirá la orden a la placa Arduino Uno conectada al mando.

La función de grabación, en cuanto la placa de la emisora este lista, comenzará a captar datos de la señal y los reenviará a la segunda placa que será la encargadade imprimirlos por el puerto serie para que *CoolTerm* los almacene en un archivo de datos *.txt*.

Para finalizar la grabación se pulsa la orden STOP que terminará la adquisición de datos de la emisora. Acto seguido se podrá disponer de los datos en el archivo de captura habilitado antes de comenzar la grabación.

La función es de reproducción, el sistema quedará esperando la carga de tramas desde *CoolTerm* que serán enviadas primero por el puerto serie y luego transmitidos a la primera placa que será la que genere la trama PPM por el pin de salida. La trama introducida se repetirá indefinidamente hasta que una nueva trama sea enviada.

Para finalizar la reproducción, al igual que en la función de grabación, se pulsa la orden STOP y se deja de generar señal.

6 CONCLUSIONES Y LINEAS FUTURAS DE DESARROLLO

Este estudio ha tenido un objetivo bastante atractivo que ha servido para motivar los avances y obtener resultados que, aun no siendo los esperados, han cumplido las expectativas planteadas al inicio del proyecto.

La animatrónica se basa en el prototipado y en la necesidad de ingeniar mecanismos y máquinas de presupuesto reducido y con unos requisitos muy concretos. Es exactamente la filosofía que ha impulsado este trabajo.

Al final de este estudio se ha desarrollado un dispositivo con una interfaz táctil capaz de capturar y guardar señales de control en tiempo real, utilizando una plataforma de bajo coste y un ordenador.

Desde el principio se han encontrado numerosas dificultades ya que, una vez estudiada la señal, se comprendió que se necesitaba trabajar con interrupciones por hardware debido a la importancia de los tiempos y los intervalos tan reducidos con los que se trabaja. No obstante se obtuvieron resultados para seguir trabajando bastante rápidos.

Tras comprender la señal y captar datos correctos de la misma se pasó a la escritura de datos en memoria, lo que ha consumido prácticamente la totalidad del tiempo de desarrollo de este proyecto. La complicación que ha supuesto intentar sincronizar las señales y los procesos de escritura en la SD ha sido una ardua tarea que ha culminado en un muestreo de la señal que permite caracterizar el proceso de control con cierta calidad. Se ha trabajado con distintas arquitecturas, plataformas y configuraciones como se muestra en los apartados de este documento.

Sin embargo, de cara a obtener una grabación más fiel de la señal se ha seguido trabajando y barajando la opción de una adquisición de datos más fiable comunicando el montaje con un ordenador mediante el puerto serie. Esto ha permitido captar una señal de mayor calidad y la posibilidad de analizar los datos rápidamente, obteniendo gráficas y monitoreando el control.

Se cierra con este documento una primera etapa en el desarrollo de este dispositivo, pero quedan multitud de aspectos a evolucionar. A continuación se exponen una serie de líneas futuras de investigación y desarrollo.

- Búsqueda de un dispositivo compacto y autónomo trabajando con otro tipo de hardware. Barajar otras posibilidades de procesadores y memorias con un mayor rendimiento para mejorar la velocidad de almacenamiento de los datos captados por el dispositivo.
- Planteamiento de nuevas configuraciones y sincronización entre dispositivos. Explorando nuevos algoritmos y tratamiento de la señal se puede optimizar el código para agilizar la grabación.
- Tratar en mayor profundidad la generación y reproducción de los patrones de control previamente grabados.
- Mejora de la interfaz de usuario de cara a un mayor control y una ampliación de las opciones de control mediante la pantalla táctil. Por ejemplo poder gestionar los archivos de grabación desde la propia pantalla, nombrar, seleccionar y borrar archivos para una mejor organización.

REFERENCIAS

[1] «Web oficial de Arduino,» [En línea]. Available: www.arduino.cc.

[2] *Software Fritzing.*

[3] «Web oficial de Geekcreit,» [En línea]. Available: www.geekcreit.com.

[4] «Web oficial de Samsung,» [En línea]. Available: www.samsung.com.

[5] *Manual de usuario de la emisora R/C Eachine i6.*

[6] «Web oficial de Eachine,» [En línea]. Available: www.eachine.com.

BIBLIOGRAFÍA Y SITIOS WEB DE CONSULTA

- *"Exploring Arduino: Tools and Techniques for Engineering Wizardry"*. Jeremy Blum
- Web oficial de Arduino - www.arduino.cc
- Web oficial de Geekcreit - www.geekcreit.com
- Web oficial de Adafruit - www. adafruit.com
- Web oficial de Eachine - www.eachine.com
- GitHub - www.github.com
- Roger Meier's Freeware - http://freeware.the-meiers.org (Software CoolTerm)
- Consulta en multitud de foros, videotutoriales y blogs de autores particulares.

ANEXO A – HOJA DE CARACTERÍSTICAS ARDUINO YÚN

ANEXO B – HOJA DE CARACTERÍSTICAS ARDUINO UNO