Conectar Arduino a Simulink. Para el equipo de prácticas del ventilador

Dpto. Ing. de Sistemas y Automática
Universidad de Sevilla

1. Hardware

Vamos a trabajar con el ventilador de una cpu de un ordenador convencional:

Figura 1: Ventilador de 4 hilos.

Como podemos ver este ventilador tiene 4 cables:[1]

1. Cable NEGRO (en la figura 1 marcado con el signo - y en la figura 3 marcado como *GND*). Este cable va conectado a la tierra (polo negativo) de la fuente de alimentación a través del GND del Arduino.

2. Cable ROJO (en la figura 1 marcado con el signo + y en la figura 3 marcado como *12V*). Este cable va al terminal positivo de la fuente de alimentación externa a través del terminal V_{in} del Arduino.

3. Cable AMARILLO, (en la figura 1 marcado con el signo *Tach* y en la figura 3 marcado como *RPM*). Tach viene de tachometer o tacómetro en español. Este cable va conectado a tierra a través de un interruptor que el ventilador cierra una vez por vuelta. Para poder cablear esta señal al Arduino, necesitaremos añadir una resistencia entre este cable y $5V$, de forma que cuando el interruptor se cierre el cable AMARILLO se pondrá a $0V$ y cuando se abra, gracias a la resistencia añadida, se pondrá a $5V$. Véase figura 2 (el interruptor estaría dentro del motor y la resistencia se la tenemos que añadir nosotros). Esta resistencia se la añadiremos por software definiendo este pin como *INPUT_PULLUP* En nuestro programa de Matlab. Véase la sección 3.3.

   Esta entrada la vamos a utilizar como una interrupción externa[2], va conectada al pin 2 del Arduino.

4. Cable AZUL, (en la figura 1 marcado con el signo *control*, y en la figura 3 marcado como *PWM*). El motor recibe una señal de control PWM para regular su velocidad. El ventilador espera una señal

[1] Pueden cambiar los colores de los cables, pero el negro siempre es GND y no cambia el orden

[2] Interrupción: Cuando ocurre un determinado evento el procesador "sale" inmediatamente del flujo normal del programa y ejecuta la función de interrupción asociada ignorando por completo cualquier otra tarea (por esto se llama interrupción). Al finalizar la función, el procesador vuelve al flujo principal, en el mismo punto donde había sido interrumpido.

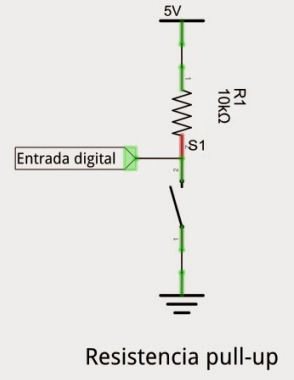

Figura 2: Resistencia pull up

que basculará de 0V a 12V, pero como todo lo que pase de una tensión umbral en torno a los $2V$ se entiende como nivel alto, el ventilador "entiende" la señal PWM que genera el Arduino, que basculará entre $0V$ y $5V$. Cablearemos pues este cable a un pin PWM (los que tienen el signo $\sim$) del Arduino. En este caso va conectado al pin 3 del Arduino.

El circuito resultante puede verse en la Figura 3:

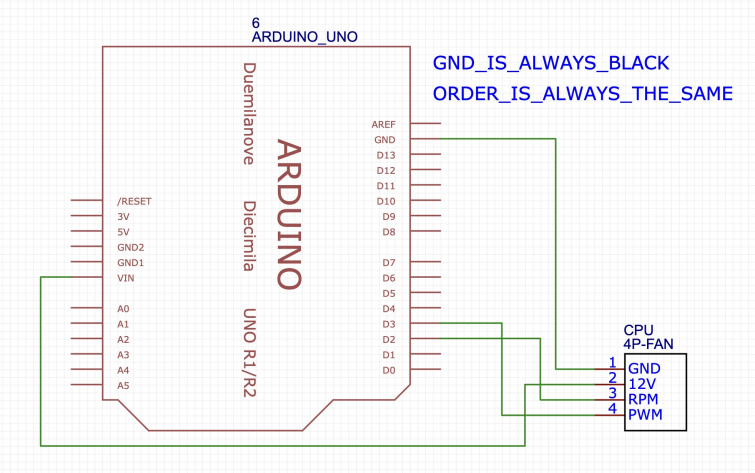

Figura 3: Circuito.

Conexión ventilador Arduino

1. Primer cable, Negro a GND del Arduino y a través del mismo al GND de la fuente.
2. Segundo cable 12V. Va directamente a los 12V de la fuente a través del pin V_{in} del Arduino.
3. Tercer cable. RPM (Suele ser amarillo). Es el cable que está conectado al encoder del ventilador. Al Pin 2 del Arduino.
4. Cuarto Cable. PWM (Suele se del color azul). Es la señal del control del Ventilador. Ha de recibir una señal PWM. Lo conectaremos al pin3 del Arduino.

2. Introducción

Hay dos maneras de trabajar con este equipo:

1. La primera es programar todo el código en Matlab usando las librerías de Arduino que habrá que instalarlas como se describe en la sección 3.1. Este procedimiento nos ahorra tener que hacer parte del código en C y tiene la ventaja de que todo el Hardware de Arduino está disponible. La desventaja es que Matlab carga su propio código en el Arduino por lo que no se pueden usar las librerías del mismo[3]. El Arduino se convierte en una mera tarjeta de adquisición de datos, muy económica.

2. La segunda manera es es usar la comunicación serie para controlar la planta. En Matlab tendremos que tener instalada la librería *Instrument Control Toolbox*, que por otro lado suele venir instalada y no da problemas. La ventaja es que tenemos a nuestra disposición todas la librerías desarrolladas para Arduino y la desventaja es que tenemos que programarnos un protocolo para enviar los datos.

   Otra ventaja es que, al correr parte del código en el Arduino, el Matlab va más fluido.

   Podemos ver como trabajar con el equipo usando el puerto serie para trasmitir la información en la sección **??**

3. Usar el equipo de prácticas usando solo Matlab y la librería de Arduino

Esta librería no viene instalada por defecto por lo que tendremos que instalarla:

3.1. Instalación de las librerías necesarias para que Matlab trabaje con Arduino

1. Abrimos Matlab[4]
2. En la barra de herramientas nos vamos a $Add-Ons$ y luego a $Get\ Add-Ons$[5].
3. Se abre c[6]
4. En la parte de arriba en el buscador escribimos: Arduino Matlab
5. Instalamos *Matlab Support Package for Arduino Hardware*
   - Una vez instalada te pide si quieres inicializarla. Para ello necesitas conectar el Arduino que se vaya a usar y seguir los pasos.
6. De nuevo en el $Add-Ons\ explorer$, en el buscador escribimos: Arduino simulink
7. Instalamos *Simulink Support Package for Arduino Hardware*
   - Una vez instalada te pide si quieres inicializarla. Para ello necesitas conectar el Arduino que se vaya a usar y seguir los pasos.
8. En *Simulink Support Package for Arduino hardware - Examples*. Podemos ver multiples ejemplos de cómo usar la librería.

3.2. Conexión Matlab con la señal PWM del Arduino que controla la velocidad del ventilador

Empezamos creando una simulación nueva en Simulink. Se necesitan los siguientes bloques:

1. Bloque PWM . Abra el *Library Browser* y busque la librería: *Simulink Support Package for Arduino ...*; dentro de *Common*, se busca el bloque PWM y lo arrastramos a la simulación.
   - Se hace doble click sobre le bloque y se cambia el pin al número 3, que es el pin de control de la velocidad del ventilador.

[3]Cualquier código que albergue el Arduino será sobreescrito.
[4]Puede instalar la última version de la página de la universidad con la licencia campus.
[5]https://www.youtube.com/watch?v=fXSQkRN-USw
[6]Si tienes una versión antigua de Matlab, tendrás que instalar la versión manualmente descargando el archivo. El propio Matlab te dará las instrucciones.

2. Bloque *Constant*. Librería *Simulink*, carpeta *Sources* se escoge el bloque *Constant* y se arrastra a la simulación.

3. Bloque *Slider Gain*. Librería *Simulink*, carpeta *Math Operators* se escoge el bloque *Slider Gain* y se arrastra a la simulación.

   - Se hace doble click sobre este bloque y ajustamos el campo *Low* a 0 y el *High* a 255.[7].

4. Bloque *Convert*. Librería *Simulink*, carpeta *Commonly Used Blocks*, se escoge el bloque *Convert* y se arrastra a la simulación.

   - Se hace doble click sobre el bloque y en *Output data type* se escoge *unit8* (ya que debe recibir un número entre 0 y 255).

5. Se conecta *Constant* con *Slider Gain*, éste a su vez, con el bloque *Convert* y este último al bloque *PWM*. Ver figura 4.

6. Se selecciona la pestaña *MODELLING* y se hace click en *Model Settings*. Se abrirá una nueva ventana de diálogo que se llama *Configuration Parameters*.

7. En en menú de la izquierda, se hace click en *Hardware Implementation*

8. En *Hardware board*, se escoge la placa, *Arduino Uno*.

9. Recomendable, en *Tarjet hardware resources*, se busca *Serial port properties*. En *Serial o baud rate*, se eleva la velocidad del puerto serie a 115200 baudios.

10. Se hace click en OK en la parte de abajo de la ventana *Configuration Parameters*.

11. Se puede observar que ha aparecido una nueva pestaña, al lado de *FORMAT*, que se llama *HARDWARE* y que está activa.

12. En *Run on board*, se escoge *Run on board (external hardware)*.

13. Se conecta el equipo de prácticas al PC mediante el cable usb y a la alimentación, mediante la fuente de alimentación. El ventilador empezará a girar.

14. En *Stop time*, se pone inf para que el tiempo de simulación sea infinito.

15. A continuación, se hace click en *Monitor & Tune*. Matlab compilará el programa y lo cargará en el Arduino. Esta tarea suele tardar un poco. Se podrá ver que se ejecuta porque aparecerá la barra de tiempo.

16. Durante la simulación se podrá cambiar el valor de la ganancia de *Slider Gain*[8] de 0 a 255. Se debe apreciar que el ventilador gira a su máxima velocidad cuando el valor que le llega al bloque PWM es de 255, y cuando el valor que le llega es 0 el ventilador girará a su mínima velocidad.[9]

Configuración

Siempre que queramos ejecutar un ejemplo o nuestro propio programa, en las opciones de *MODELLING* → *Model Settings*

1. Buscar la opción *Hardware Implementation*
2. En **Hardware board:** Escogemos nuestra placa, Arduino Uno.
3. (recomendable) En *Tarjet hardware resources* buscamos *Serial port properties*. En serial *Serial o baud rate* elevamos la velocidad del puerto serie a 115200 baudios.

[7] El controlador de la señal PWM del Arduino es de 8 bits. Puede tomar valores comprendidos entre 0 y 255 ($2^8 - 1$). con una precisión de 1.

[8] Haciendo doble click sobre el bloque.

[9] Hay que tener cuidado de no darle señales de control fuera de ese intervalo o nuestro ventilador empezará a hacer cosas raras.

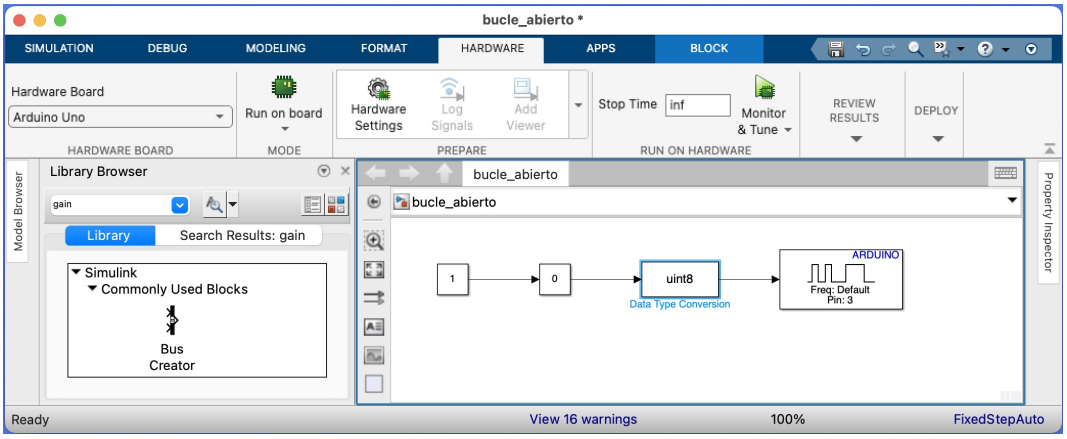

Figura 4: Control de la velocidad en bucle abierto.

Ejecución

Una vez configurada la comunicación con el Arduino:

1. En la pestaña *HARDWARE*
2. En *Run on board*, escogemos *Run on board (external hardware)*.
3. A continuación hacemos click en *Monitor & Tune*. Matlab compilará el programa y lo cargará en el Arduino. Esta tarea suele tardar un poco. Podremos ver que se ejecuta porque aparecerá la barra de tiempo.

3.3. Conexión Matlab con la señal del encoder del ventilador

Para leer el valor del encoder y puesto que este cambia muy rápidamente, vamos a usar una interrupción. Es decir, cada vez que el encoder cambie de valor haremos que el programa principal se pare y ejecute una rutina de interrupción (haremos que se incremente el valor de un contador). Después de ejecutarla seguirá ejecutando el programa principal donde lo dejó.

Se empieza creando una simulación nueva en Simulink. Se necesitarán los siguientes bloques:

1. Bloque *External Interrupt*. Se abre el *Library Browser*, busque la librería: *Simulink Support Package for Arduino ...*, dentro de *Common*, se busca el bloque *External Interrupt* y se arrastra a la simulación. Se hace doble click sobre le bloque y se cambia:
   - El pin del Arduino nos va a generar la interrupción, en nuestro caso el 2.
   - Se selecciona *Pull-up Input Pin*. Para que el Arduino añada por software una resistencia de Pull-up. Se vio porqué esta resistencia era necesaria en la sección 1.
   - En *Mode* se cambia de RISING (detecta cuando la señal cambia de 0 a 1) a CHANGE para que se detecte cuado cambia de 0 a 1 y de 1 a 0 (se conseguirá una mejor precisión de la medida de la velocidad).
2. Bloque *Function-Call Split*. Librería *Simulink*, carpeta *Ports & Subsytems*, se escoge el bloque *Function-Call Split* y se arrastra a la simulación.
3. Se conecta la salida *IRQ* del bloque *External Interrupt* a la entrada *function()* del bloque *Function-Call Split*. Este bloque se ejecutará cada vez que el bloque *External Interrupt* detecte un cambio en la señal conectada al pin2 del Arduino.
4. Se va a hacer un código que cada vez que se ejecute el bloque anterior se incremente un número. Para ello se hace click 2 veces sobre el bloque y creamos el siguiente programa en el interior (figura 5):

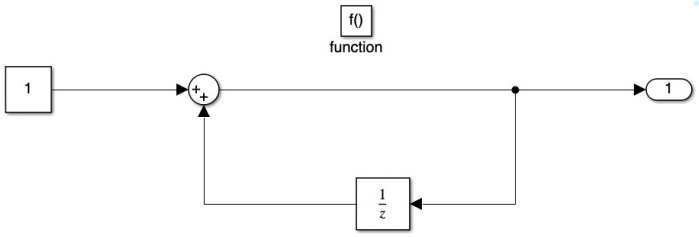

Figura 5: Contador.

Para crear este programa hacen falta los siguientes bloques:

- Bloque *Constant*. Librería *Simulink*, carpeta *Sources* se escoge el bloque *Constant* y se arrastra a la simulación.
- Bloque *Sum*. Librería *Simulink*, carpeta *Commonly Used Blocks* se escoge el bloque *Sum* y lo arrastramos a la simulación.
- Bloque *Delay*. Librería *Simulink*, carpeta *Discrete* se escoge el bloque *Delay* y se arrastraa la simulación. Se hace doble click sobre él.
  - El campo value, se pone 1 (el bloque recordará el valor de la entrada del 1 ciclo atrás).
- Una vez fuera de la ventana de diálogo, se selecciona el bloque *Delay* y se le da 2 veces a Control+R (rotará 180 grados).
- Construimos el diagrama de la figura 5. Cada vez q se ejecute la función, puesto que el bloque delay "recuerda" el valor de la llamada anterior, hará que la salida se incremente en 1.

5. Configuración de la simulación: *MODELLING* → *Model Settings*
   - En la pestaña *Hardware implementation*, en *Hardware board:*, se selecciona Arduino Uno.
   - Recomendable, en *Tarjet hardware resources* se busca *Serial port properties*. En *Serial o baud rate* se eleva la velocidad del puerto serie a 115200 baudios.
   - En el menú de la izquierda, arriba se seleccionaSolver. En *Stop time* se pone inf para que el tiempo de simulación sea infinito.
   - En *Solver Selection*, dentro de campo *Type* se seleciona *Fixed-step*
   - Hacemos click en *Solver details*
   - En el campo *Fixed-step size (fundamental sample time):*, se pone 0,1, porque el tiempo de muestreo del ventilador es $100ms$.
   - Se hace click en OK (abajo).
6. El periodo de muestreo de la simulación en $0{,}1s$. Sin embargo, el bloque *External Interrupt* generará datos cuando haya cambios en la señal asociada al encoder, es decir a una frecuencia distinta al periodo de muestreo. Necesitamos un bloque adapte las frecuencias de muestreo. Este bloque se llama *Rate Transition* y se encuentra en:
   Librería *Simulink*, carpeta *Signal attributes* se escoge el bloque *Rate Transition* y se arrastra a la simulación. Se hace doble click sobre el bloque:
   - Se deselecciona *Ensure deterministic data transfer (maximun delay)*, ya que los datos de entrada no vienen con una frecuencia fija.
   - En *Output port sample time options* se pone *Inherit* para que utilize el tiempo de muestreo de la simulación.

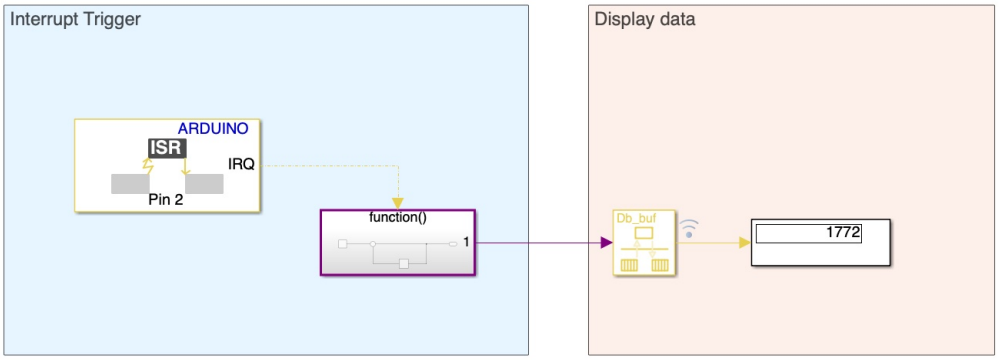

Figura 6: Bloques necesarios para leer el encoder del ventilador.

7. Se utilizará el bloque *display* para ver el resultado. Librería *Simulink*, carpeta *Sinks* se escoge el bloque *display* y se arrastra a la simulación. SeConectamos los bloques como en la figura 6.

8. Antes de lanzar la simulación, en la pestaña HARDWARE

   - En *Run on board*, se escoge *Run on board (external hardware)*.
   - A continuación, se hace click en *Monitor* & *Tune*[10]. Matlab compilará el programa y lo cargará en el Arduino. Esta tarea suele tardar un poco. Podremos ver que se ejecuta porque aparecerá la barra de tiempo.

Podrá apreciarse en el display, que aparece un número que se va incrementando con el tiempo. Es el número de cambios de la señal del encoder. Se quiere reiniciar ese contador cada periodo de muestreo, para ello se añade al código un retraso[11] y un restador[12] (figura 7).

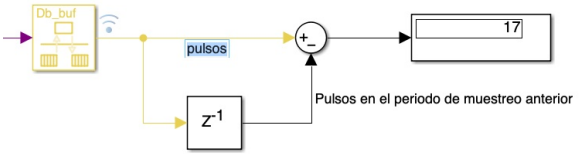

Figura 7: Pulsos cada 100ms.

El número de cambios de la señal del encoder, muestreado cada $100ms$, saldrá por la salida del bloque *Rate transition*. Por la salida del bloque *Unit Delay* sale el valor de esa misma señal en el periodo de muestreo anterior. Por lo tanto por la salida del circuito restador saldrán los pulsos dados los últimos $100ms$.

Si ahora añadimos a nuestra simulación el código que hicimos en la sección 3.2 podremos medir los cambios de la señal del encoder cada $100ms$ para cualquier velocidad. Ver figura 8

IMPORTANTE

Si se ejecuta la simulación de la pestaña SIMULATION no funcionará. Hay que lanzarla desde HARDWARE para que se cargue el programa apropiado en el Arduino.

[10] No olvide tener conectado el Arduino al PC.
[11] En la librería Simulink → Discrete → Unit Delay.
[12] En la librería Simulink → Commonly Used Blocks → Sum. Podemos convertirlo en restador modificando sus propiedades.

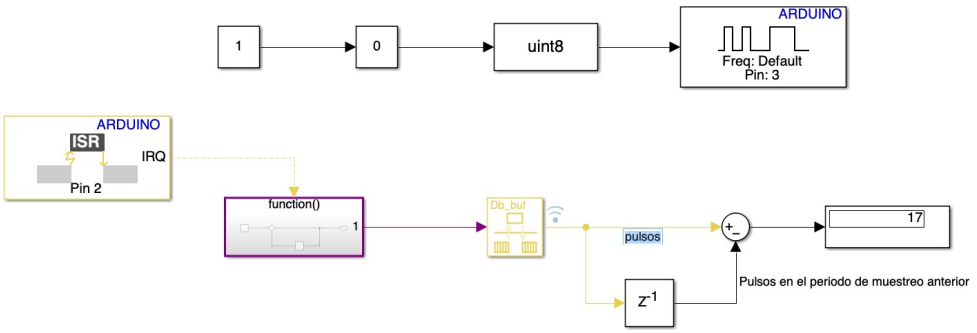

Figura 8: Pulsos cada 100ms controlados por la señal PWM.

3.4. Programa en Matlab que mide la velocidad del ventilador

El rango de medidas que se obtiene es muy pobre. En el caso del equipo con que se diseño la práctica se obtienen valores que van de 5 a 17 de uno en uno.

Una posible manera de enriquecer la medida es sumar las 10 últimas medidas obtenidas. De esta forma estaríamos midiendo los cambios de la señal del encoder (que es proporcional a la velocidad) cada segundo (ya que se toma una medida cada décima de segundo), pero actualizando el valor cada $100ms$. El inconveniente es que añadimos cierto retraso a la medida, es decir, desde que hacemos un cambio en la señal PWM, hasta que obtenemos la medida definitiva de la velocidad, se tarda cierto tiempo. La forma de programarlo es muy sencilla si usamos el bloque *delay*[13]. Si hacemos doble click sobre el bloque para ver sus parámetros veremos que nos deja configurar el retraso en periodos de muestreo (parámetro *value*). Si se copian 9 de estos bloques conectados a la señal de la velocidad y configuramos cada uno con un retraso de 1 a 9, tendremos las 9 últimas medidas de los pulsos, con lo que solo nos quedará sumarlas a la actual (ver figura 9) para tener el número de cambios en la señal del encoder cada segundo. Podemos ver que también:

- Se ha añadido a la señal PWM una saturación[14] entre 0 y 255, para que no lleguen señales fuera de rango al Arduino.
- A la señal de salida se le ha añadido un bloque *convert*[15] para convertir el dato (cambios de la señal asociada al encoder por segundo) a *double* que es con la precisión que trabaja Matlab.
- Por último, se ha añadido una ganancia[16] de valor 0,5 para convertir el número de cambios por segundo de la señal de encoder (el encoder se cierra y abre una vez por vuelta) en revoluciones por segundo.

En el caso del equipo de prácticas que se uso para redactar esta memoria el rango de la variable revoluciones por segundo (*rps*) es de 24.0 a 85.5 con una precisión de 0.5.

Para terminar, podemos seleccionar toda el área azul, darle al botón derecho del raton y crear un subsistema. De esta manera todo el código desarrollado para la comunicación con el ventilador quedará encapsulado en un solo bloque, que recibe un señal que va de 0 a 255 (PWM) y saca la medida de la velocidad en revoluciones por segundo.

3.5. Programa en Matlab para controlar la velocidad del ventilador

4. Ficheros STL para imprimir una carcasa

Se han diseñado una base y una tapa para albergar un ventilador de las dimensiones apropiadas y un Arduino Uno aunque no es necesario para el equipo.

Los ficheros se pueden encontrar en la misma carpeta de Github que este documento.

[13]Librería Simulink → Discrete → Delay.
[14]Librería Simulink → Discontinuities → Saturation.
[15]Librería Simulink → Commonly Used Blocks → Data Type Conversion.
[16]Librería Simulink → Commonly Used Blocks → Gain.

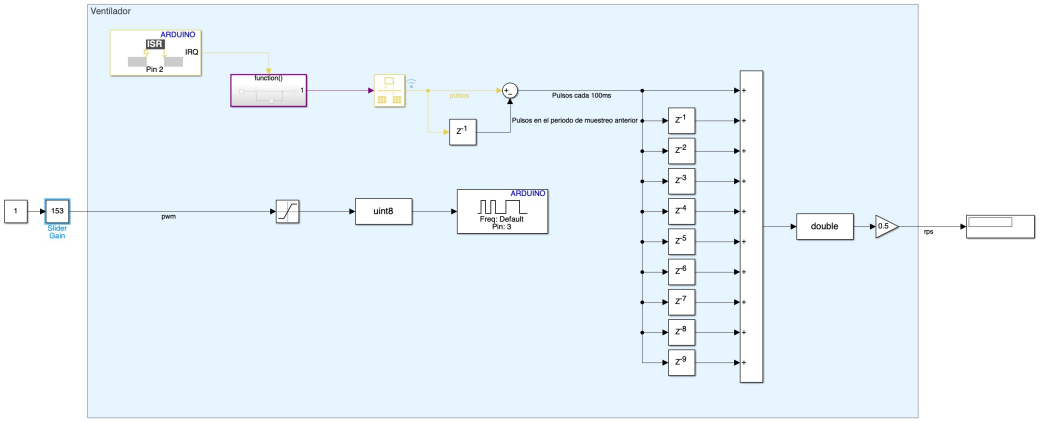

Figura 9: Programa para controlar el ventilador.

Figura 10: Bloque de Simulink de comunicación con el ventilador.

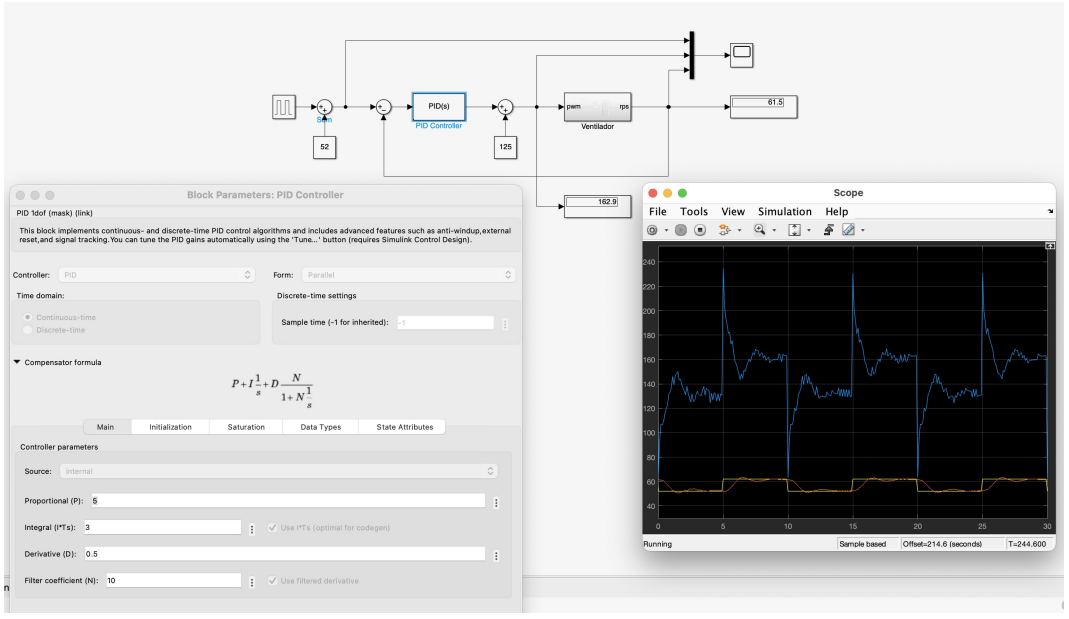

Figura 11: Sistema controlado con un PID.

5. PCB con el conexionado

Debido a la simplicidad del circuito no es necesario el uso de una pcb. Aun así se ha diseñado una para la simplificar del montaje. El fichero se encuentra en la misma carpeta que este pdf

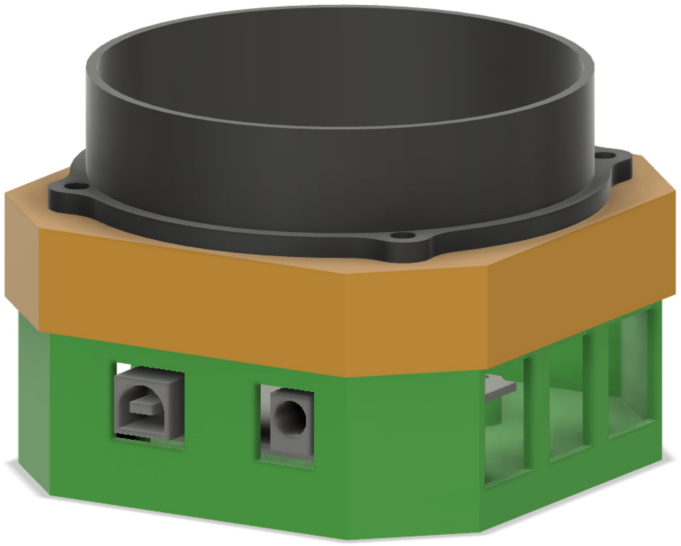

Figura 12: Aspecto del sistema montado.