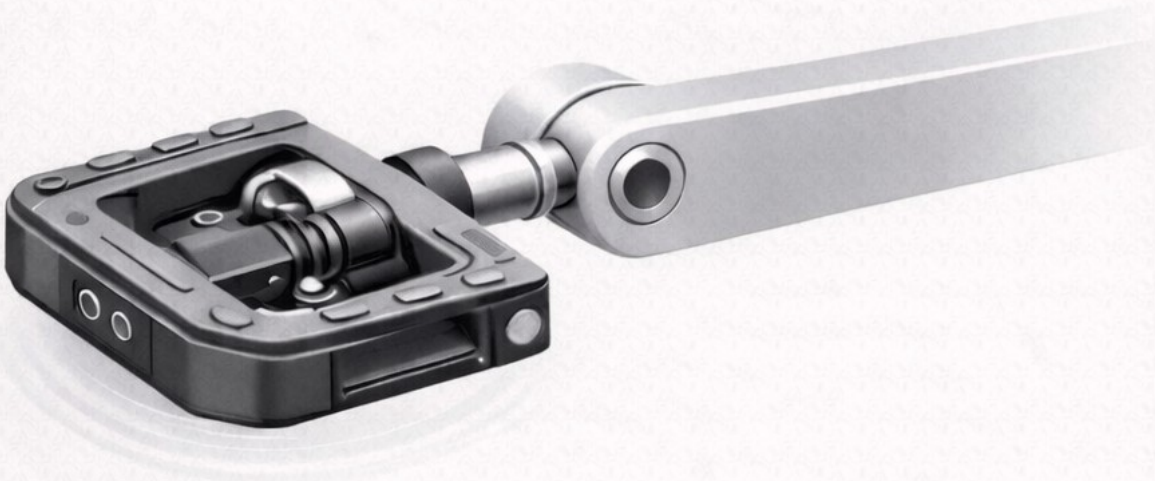
MEDIDOR DE POTENCIA MECÁNICA DE PEDALEO



Javier Narváez Ferri

Prof. Joan Masdemont Fontás

MEDIDOR DE POTENCIA MECANICA

Objetivo: diseñar un medidor de potencia mecánica de pedalada para mejorar las prestaciones de una bicicleta de *spinning* de gama baja.

Requisitos:

a) Medir la velocidad angular de giro de la rueda.
b) Medir o estimar la fuerza aplicada al pedal.
c) Mostrar en una pantalla TFT la velocidad lineal, la potencia absoluta en watios, y la potencia relativa respecto a un valor de referencia de 150 W.

Este proyecto se ha realizado con asistencia de inteligencia artificial. Los códigos desarrollados para este proyecto en Python y en C++, se pueden consultar en el siguiente repositorio de Github:

https://github.com/javnarv/Power-meter-for-spinning-cycling

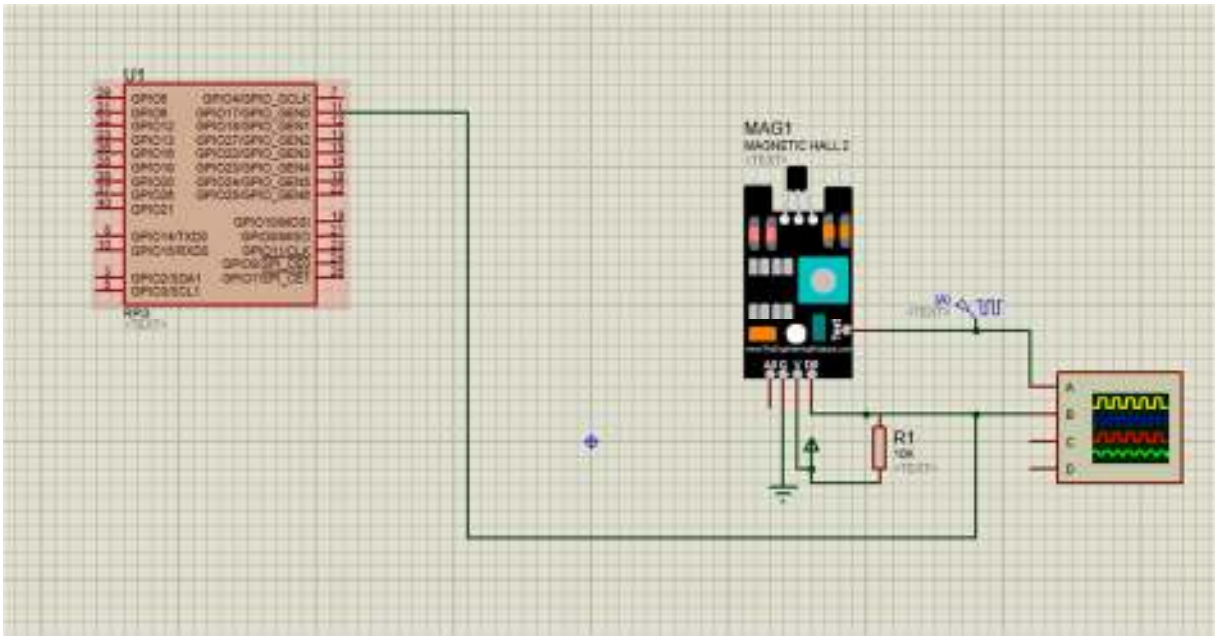
1. Medición de la velocidad angular

Para medir la velocidad angular empleamos un sensor de efecto Hall. Para que la lectura del giro de la rueda de inercia sea estable, debe seleccionarse un sensor que tenga efecto *"latching"*, es decir que se active cuando uno de los polos del imán pase cerca del sensor, y que permanezca activado hasta que pase un imán de polaridad contraria.

Se ha seleccionado el modelo US1881, que se activa con el polo sur de un imán y se desactiva con el polo norte de otro imán. Igualmente se han seleccionado dos imanes comunes, de los que se emplean en los cierres de puertas, que se colocaran por simple acople magnético sobre la rueda de inercia (ver foto).

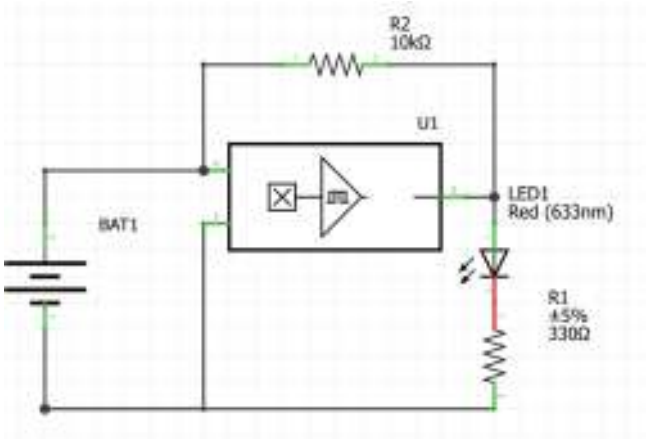
Se ha desarrollado un software en Python siguiendo un modelo de interrupción, es decir que se produzca un evento cada vez que se active el sensor, en cuyo caso la Raspberry recibirá un flanco de subida en el GPIO seleccionado, en este caso el número 17[1].

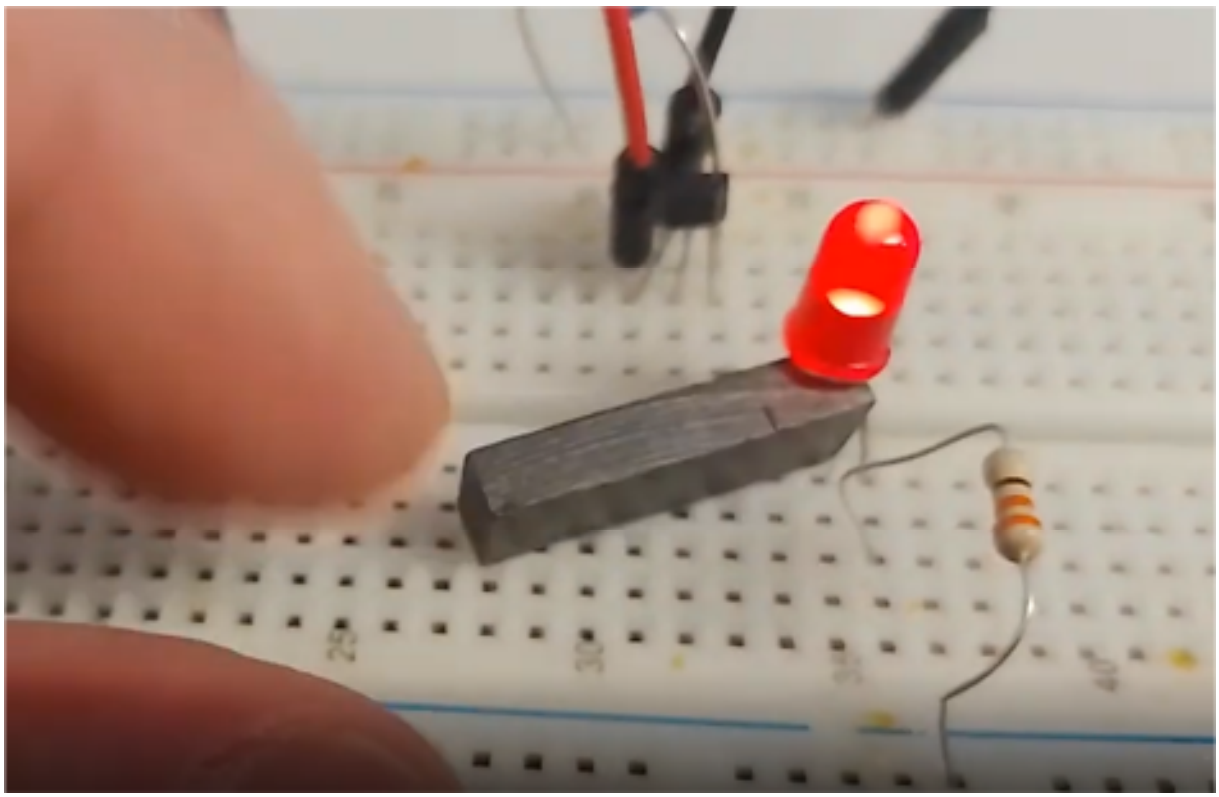
Para probar el software, se ha realizado una simulación en Proteus:



El dispositivo de efecto Hall incorporado en la librería de Proteus tiene una salida analógica, que no utilizamos, una salida digital, que conectamos al GPIO 17 de la Raspberry, una entrada de Vcc, una entrada de Gnd, y una entrada de test, al que conectamos un generador de pulsos, de 1.5 Hz de frecuencia, para simular el efecto de los imanes sobre el sensor. También acoplamos un osciloscopio para ver los pulsos de entrada y la salida digital del sensor, y una resistencia de *pull-up*.

Para probar el sensor US1881 y los imanes, montamos un circuito sencillo en una *protoboard*, alimentándolo a 5 voltios, y añadiendo un led y una resistencia de 330 Ω en el pin de salida.





Vemos que, efectivamente, cuando se acerca el polo sur del imán a la parte plana del sensor, éste se activa y el led se enciende, permaneciendo activo hasta que se acerca el polo norte del imán. También se establece que la distancia entre el sensor y el imán debe ser inferior a 25 mm para que se produzca la activación. Este dato es fundamental a la hora de situar el sensor sobre el chasis de la bicicleta.

Se ejecutó el código en la Raspberry de forma satisfactoria: el código detectaba perfectamente los flancos de subida de la señal de salida del US1881 cuando se activaba manualmente con el imán. Sin embargo, tras una actualización del sistema requerida por el código de la TFT, el programa dejó de funcionar: en *runtime* daba error con el mensaje *“Failed to add edge detection”*.

Tras investigar en los foros de Raspberry[2], el problema parecía motivado por la librería del GPIO, por lo que, siguiendo el consejo del foro, la sustituí por la librería LGPIO, con los siguientes scripts:

```
sudo apt remove python3-rpi.gpio
sudo apt update
sudo apt install python3-rpi-lgpio
```

con lo que el problema se solucionó.

Llegados a este punto, procedí a realizar una prueba sobre la bicicleta de spinning, situando dos imanes sobre la rueda de inercia, y el sensor en la parte inferior del chasis, de forma que se activara y desactivara cuando pasaran los imanes respectivos.
Tras la primera prueba, me di cuenta que había un error de concepto en el programa, puesto que calculaba la velocidad en rpm contando el número de pulsos recibidos y dividiendo por el tiempo empleado. En realidad, los rpm a los que estamos habituados se refieren a la frecuencia de pedaleo, no a la frecuencia de giro de la rueda. Debido a los engranajes, cada vuelta de los pedales se traduce en 4 vueltas en la rueda de inercia. Procedí a modificar el programa añadiendo una división por cuatro en la fórmula original que calculaba los rpm[3].
Realicé varias pruebas dando exactamente 100 pedaladas (contando un solo pie) a un ritmo constante, y contrastando los valores de velocidad y de distancia que reflejaba la consola original de la bicicleta y los datos grabados en el fichero de bitácora que escribía la Raspberry [4]. Dado que no coincidían mucho, modifique la estructura del fichero de bitácora para que grabara también el número de pulsos detectados. Continuando con las pruebas de 100 pedaladas, observé que se reflejaban valores comprendidos entre 500 y 700 pulsos, cuando el valor teórico es de 400. Estos datos indicaban que el US1881 estaba generando más pulsos que los debidos. La solución que se me ocurrió fue aumentar el *bouncetime* en la llamada a la función de detección de pulsos, que estaba en 50 ms[5]. El *bouncetime* es un tiempo de latencia durante el cual no puede volver a activarse la llamada *callback* a una función. Para establecer el nuevo valor calculé que para una frecuencia de pedaleo de 100 rpm se debían producir 400 activaciones del sensor por minuto, es decir una cada 150 ms, por lo que decidí establecer el *bouncetime* a 100 ms.
Repetida la prueba de las 100 pedaladas, el fichero de bitácora reflejó 398 pulsos detectados, por lo que lo consideré aceptable. Respecto a la

distancia, la Raspberry refleja una distancia mayor que la que registra la consola original, que en todas las pruebas de 100 pedaladas ha sido de 450 metros. Haciendo el cálculo, eso supondría que el diámetro de la rueda de inercia debería ser 358 mm, en vez de los 420 mm que he medido. Obviamente, también la velocidad en Km/h que refleja la consola original es menor al valor calculado por la Raspberry. En principio, creo que mis cálculos son más correctos.

2. Medición de la fuerza

Tras evaluar diferentes alternativas, y para minimizar las intervenciones mecánicas en la bicicleta, se ha optado por medir directamente la fuerza en el pedal, situando una célula de carga de compresión en la parte inferior del pedal.

El pedal mide 90x75 mm (ver foto). Por motivos de disponibilidad se ha seleccionado una célula de carga de 50 Kg y 3 hilos. Esta célula tiene unas dimensiones de 35x35 mm, por lo que cabe perfectamente en el pedal. La biela de esta bicicleta tiene una longitud de 0.17m, por lo que el par máximo sería de 83.3 Nm.

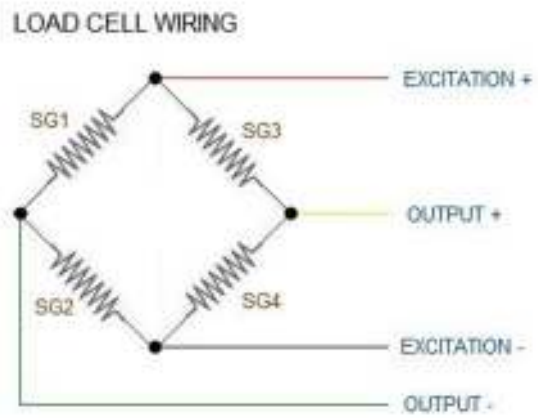
En una sesión de *spinning* simulando una subida fuerte se puede considerar una velocidad típica de 70 rpm. La conversión a radianes por segundo vendría dada por la siguiente fórmula:

$$w\left(\frac{rad}{s}\right) = 2 * \pi \; x \; rpm/60$$

Por tanto, esta célula nos podría facilitar una medida de más de 600 W en condiciones de intensidad elevada, siempre que se pedalee sentado.

2.1. Instalación de la célula de carga

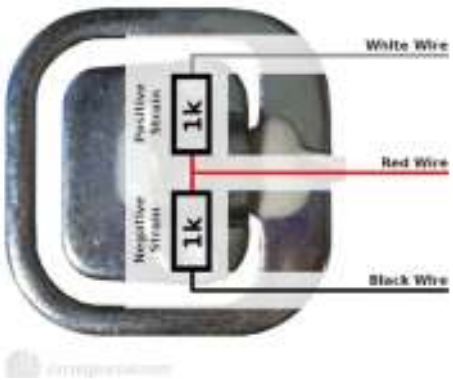
Una galga extensiométrica es un dispositivo que modifica su resistencia eléctrica de manera lineal cuando está sometida a una deformación mecánica. Las células de carga de 3 hilos están configuradas por dos galgas extensiométricas. La resistencia de ambas galgas en ausencia de torsión está alrededor de 1 KΩ.



El primer paso para la instalación es determinar el punto de unión de ambas galgas, normalmente conectado con un cable de color rojo (ver figura). Midiendo con el *tester*, se debe comprobar que la resistencia entre el cable rojo, y cualquiera de los otros dos cables esté alrededor de 1 KΩ. Consecuentemente la resistencia entre el cable blanco y el cable negro, debe estar en torno a los 2 KΩ.

Para la aplicación se utiliza la configuración conocida como puente de Wheatstone, que consiste en colocar 4 resistencias formando un rombo:

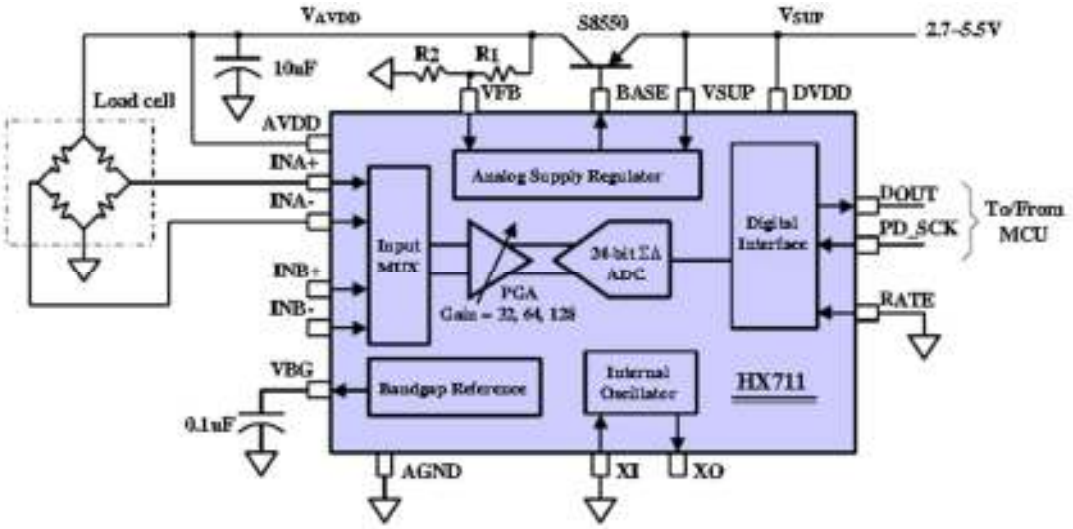
Alimentando los extremos del rombo a +Vcc y a GND, respectivamente, y conociendo los valores de tres de las resistencias, se puede deducir fácilmente el valor de la otra midiendo la diferencia de tensión en los puntos medios del puente (“*Output+” y “Output-“*) aplicando la ley de Ohm y las leyes de Kirchoff.



En el caso de nuestra célula de carga, se forma el puente de Wheatstone añadiendo externamente dos resistencias de 1 KΩ. Si colocamos la celda en el brazo derecho del puente, y las dos resistencias externas en el brazo izquierdo, se producirá una variación en la tensión medida en el punto "*Output+"* cuando se aplique un peso a la célula. La tensión en el punto "*Output-"* permanecerá constante. Por tanto, la diferencia de tensión entre los dos puntos "*Output"* variará linealmente en función del peso aplicado a la célula. Un punto importante es que, en ausencia de peso, esta diferencia de tensión debe ser lo más próxima a cero posible, por lo que los valores de las resistencias externas deben ser lo más homogéneos posibles, por lo que es conveniente emplear resistencias comerciales del 1% de variación, mejor que las del 5%.

De todas maneras, la diferencia de tensión es pequeña, del orden de decenas de mV, por lo que hay que amplificarla antes de poder ser utilizada como señal, y digitalizada, si se requiere procesamiento. Para ello se emplea comúnmente el convertidor A/D HX-711.

Este integrado amplifica la débil señal del puente, lo digitaliza con 24 bits (en complemento a 2) y lo envía en ráfaga a un procesador.



En esta figura se representa un circuito típico de aplicación de este integrado. Dispone de dos canales analógicos de entrada, el canal A cuya ganancia se puede programar a 128 o a 64, y el canal B cuya

ganancia está fijada a 32. Normalmente en estas aplicaciones, dado que la señal analógica de entrada es tan pequeña, se emplea en canal A con la ganancia máxima de 128, que es el valor que está seleccionado por defecto. . La frecuencia de muestreo se determina por el valor de la entrada RATE (pin 15). Si está colocada a masa, la frecuencia de muestreo es de 10 muestras por segundo, mientras que si está a VCC, es de 80 muestras por segundo. Habitualmente se controla la transferencia de datos mediante una entrada externa de pulsos proveniente de una CPU, que se aplica a la entrada PD_SCK (pin 11). Los datos digitalizados se llevan al exterior por la salida DOUT (pin 12). Esta salida normalmente está en valor alto y se pone en valor bajo cuando hay datos disponibles (cada 100 ms o cada 12.5 ms, según sea el valor de RAT). Si se aplican entonces pulsos a la entrada PD_SCK , los 24 bits de datos van siendo extraídos por la salida DOUT, empezando por el bit más significativo. Transcurridos los 24 pulsos, la señal externa debe proporcional un pulso adicional que restituya la salida DOUT a valor alto, permitiendo un nuevo muestreo de la señal analógica.

Si el chip recibe 26 pulsos en vez de 25 interpreta que en el próximo muestreo debe seleccionar la entrada analógica B con una ganancia de 32. Si recibe 27, interpreta que en el próximo muestreo debe seleccionar el canal A, pero con una ganancia de 64:

Nº pulsos	Canal	Ganancia
25	A	128
26	B	32
27	A	64

El cronograma siguiente ilustra el protocolo de extracción de datos y de asignación de canales y ganancias [a] .

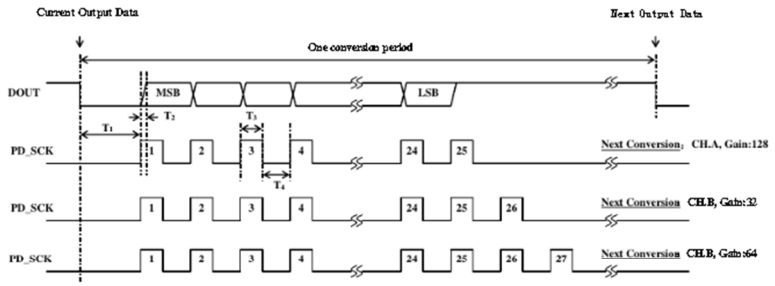


Fig.2 Data output, input and gain selection timing and control

Symbol	Note	MIN	TYP	MAX	Unit
T_1	DOUT falling edge to PD_SCK rising edge	0.1			μs
T_2	PD_SCK rising edge to DOUT data ready			0.1	μs
T_3	PD_SCK high time	0.2	1	50	μs
T_4	PD_SCK low time	0.2	1		μs

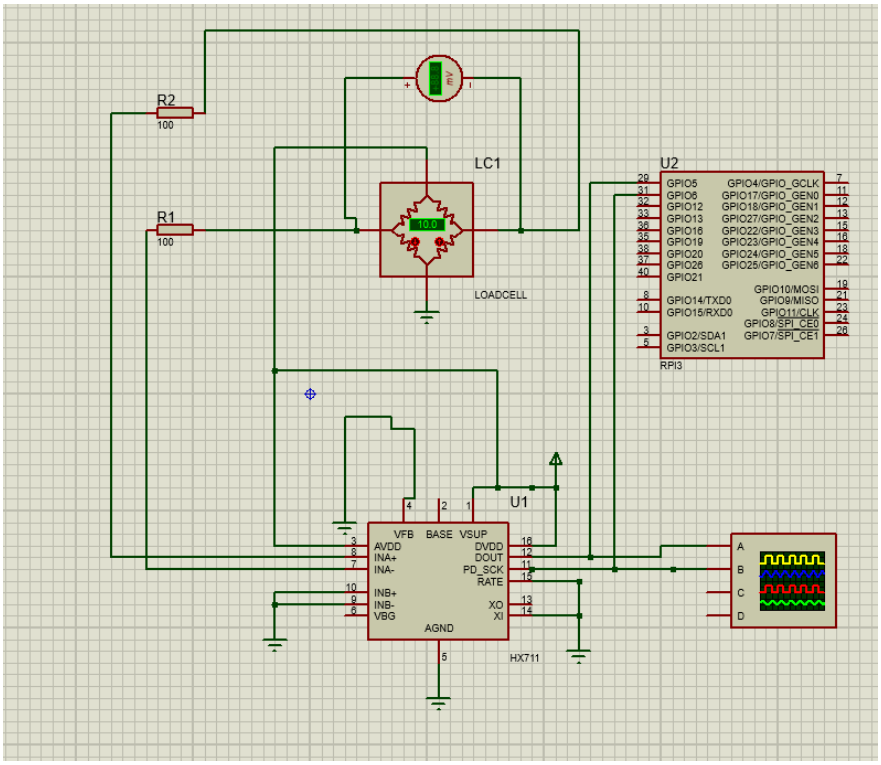
En la tabla se expresa un dato muy importante: la entrada PD_SCK no puede permanecer en valor alto más de 50 μs. El motivo es que está entrada también se emplea para inhabilitar el chip. Si sobrepasa esos 50 μs, el chip entra en estado de *“power down”* . Cuando esta entrada retorna al valor bajo, el chip vuelve a estar plenamente operativo. Este requisito tiene sus consecuencias en la programación, como veremos más adelante.

[a] Hay un error en el cronograma, el tren de bits inferior pertenece al canal A con ganancia de 64

Para la aplicación hemos escogido un módulo muy popular comercialmente, que dispone del integrado y del circuito de aplicación:

Por una banda, tenemos las 4 entradas analógicas de los canales A y B, y las dos salidas de excitación E. Conectamos las dos entradas A y las dos salidas E directamente al puente de Wheatstone, y dejamos las dos entradas del canal B sin conectar. Por la otra banda tenemos las entradas de alimentación VCC y GND, la entrada de reloj SCK y la salida de datos serie DT.

Previamente al montaje físico, realizamos una simulación en Proteus, utilizando los modelos del HX711 y de la célula de carga disponibles.

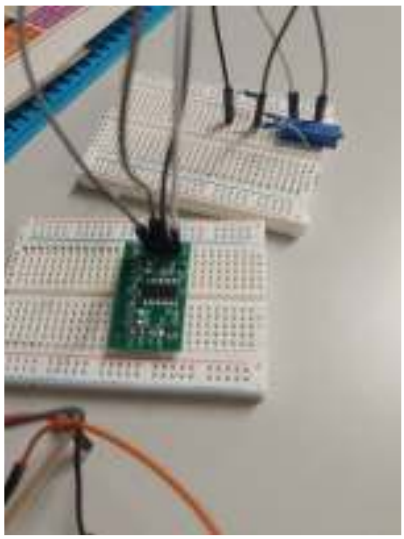


El modelo de célula de carga disponible es de 4 hilos, que es un tipo de célula que dispone de 4 galgas extensiométricas, que forman un puente de Wheatstone completo, por lo que no hay que añadir las dos resistencias de 1 KΩ exteriores.

Para hacer la simulación, creamos un código en Python, con una clase y unas funciones específicas para el hx711 [6] . La simulación funcionaba correctamente si el valor del sensor estaba a cero antes de arrancarla. Si tenía un valor diferente, no funcionaba. El problema se resolvió en la simulación anulando la función inicial de tara [b] .

Una vez comprobado el código en la simulación, procedimos a conectar la banda izquierda del módulo hx711 al puente de Wheatstone, y la banda derecha a la Raspberry: la entrada SCK al GPIO 6, la salida DO al GPIO 5. Para alimentar el módulo utilizamos las salidas de 5 V (pin1) GND (pin 3) de la Raspberry.

Modificamos el código para restaurar la función de tara, y lo ejecutamos en la Raspberry. Obtuvimos unos resultados anómalos, totalmente aleatorios, y sin influencia de la presión ejercida sobre la célula de carga.



Para analizar mejor el problema, decidimos sustituir la célula de carga en el puente de Wheatstone por una resistencia de 1 KΩ y un potenciómetro multivuelta de 5 KΩ.

Tras ajustar el puente a una tensión diferencial de 0 V, volvimos a ejecutar el programa, con los mismos resultados decepcionantes.

[b] Ver línea 106 del código

Tras analizar la situación, deducimos que el problema estaba en el estricto *"timing"* de señal de reloj que necesitaba el hx711. Nos pusimos a investigar en Internet si había ya alguna librería que solucionara este problema. Encontramos en *github* una librería en Python desarrollada específicamente para el hx711 [7] por el usuario Tatobari. En consecuencia, modificamos el código para utilizar la clase HX711 y las funciones de esta librería [8].

Ejecutamos este nuevo código con el puente de 3 resistencias y el potenciómetro conectados al módulo hx711. El resultado fue igualmente decepcionante, porque siempre daba cero, independientemente del valor del potenciómetro.

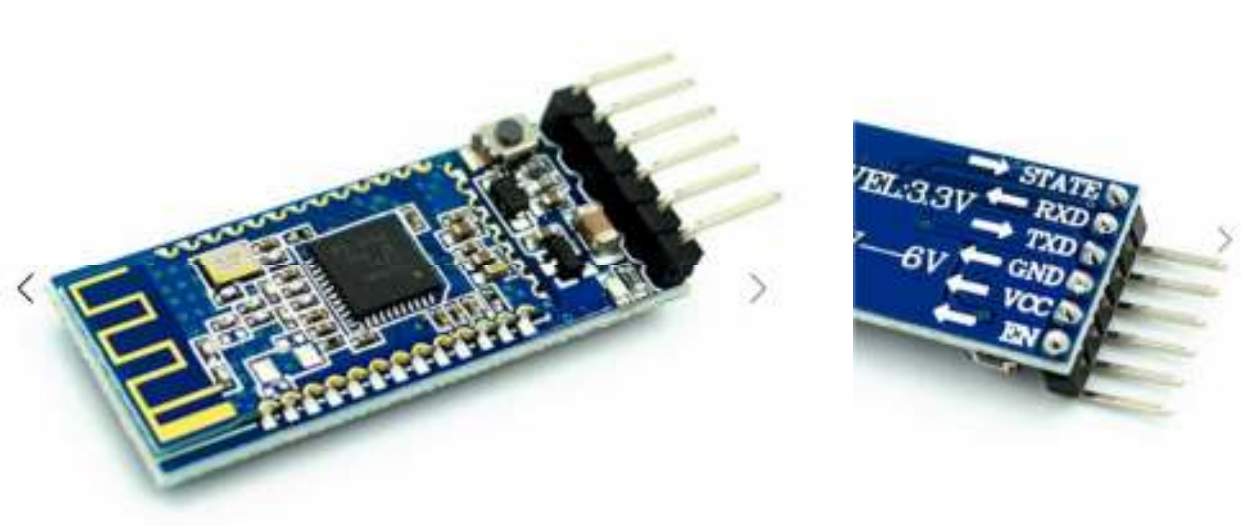
Llegados a este punto, y dado que el mismo Tatobari recomendaba, dadas las características de la Raspberry, no conectarla directamente al módulo hx711, sino utilizar un microcontrolador de puente que controlara el hx711 y enviara los datos a la Raspberry, decidimos probar el control del módulo con un Arduino. Descargamos de *github* una librería para el hx711 en Arduino [9] y realizamos un nuevo código en C++ . [10]

Esta vez el resultado fue satisfactorio, y se pudo comprobar el efecto que las variaciones del valor del potenciómetro tenían sobre el valor medido en el Arduino.

Para facilitar la manipulación de la célula de carga en las siguientes pruebas, decidimos imprimir en 3D un soporte. Encontramos en Thingiverse [11] un diseño adecuado, y lo mandamos a imprimir en el departamento de impresión 3D del centro.

2.2. Conexión Bluetooth

El siguiente paso consistió en conectar el Arduino con la Raspberry por BLE (*"Bluetooth Low Energy"*) utilizando el transmisor-receptor AT-09.

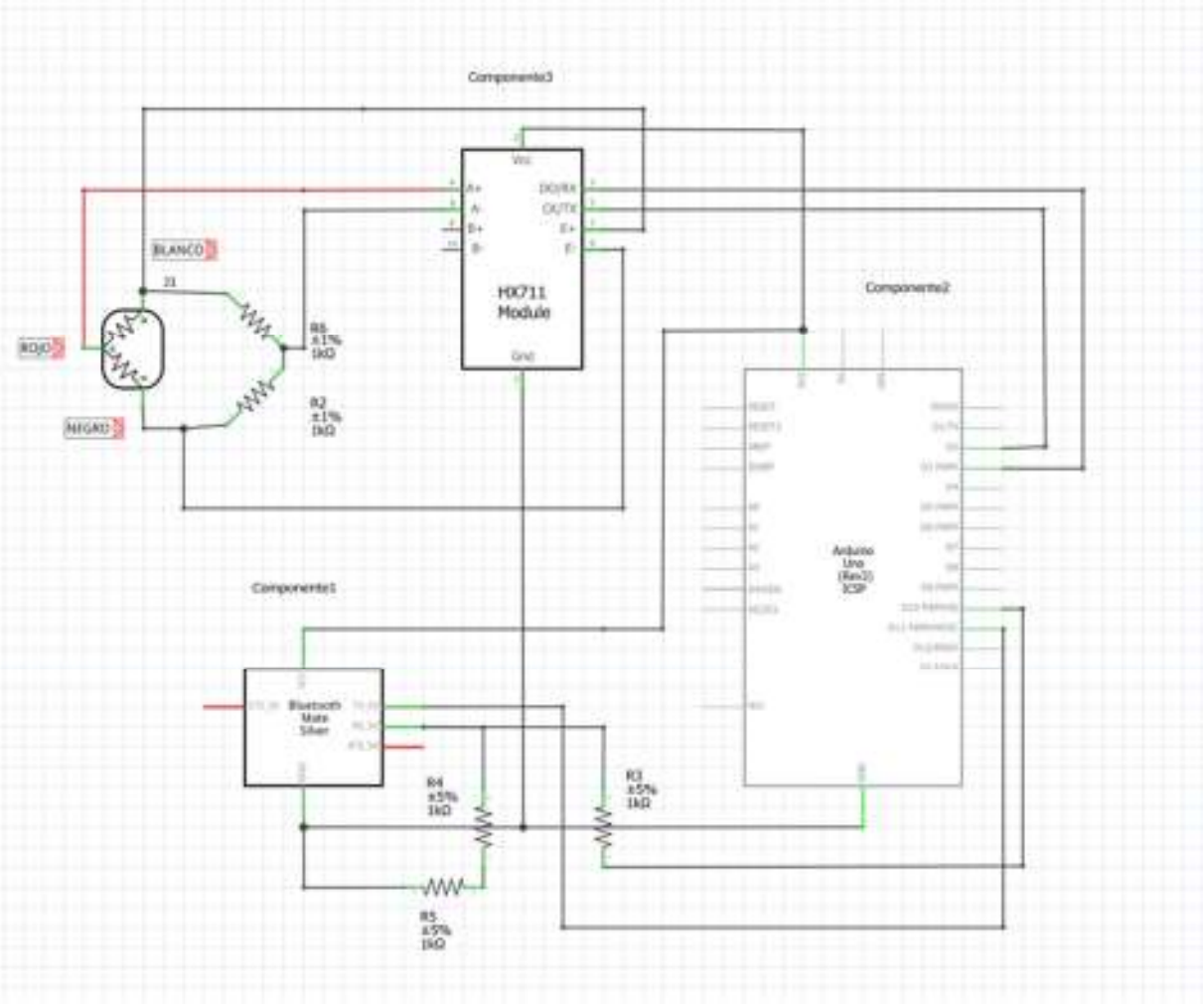


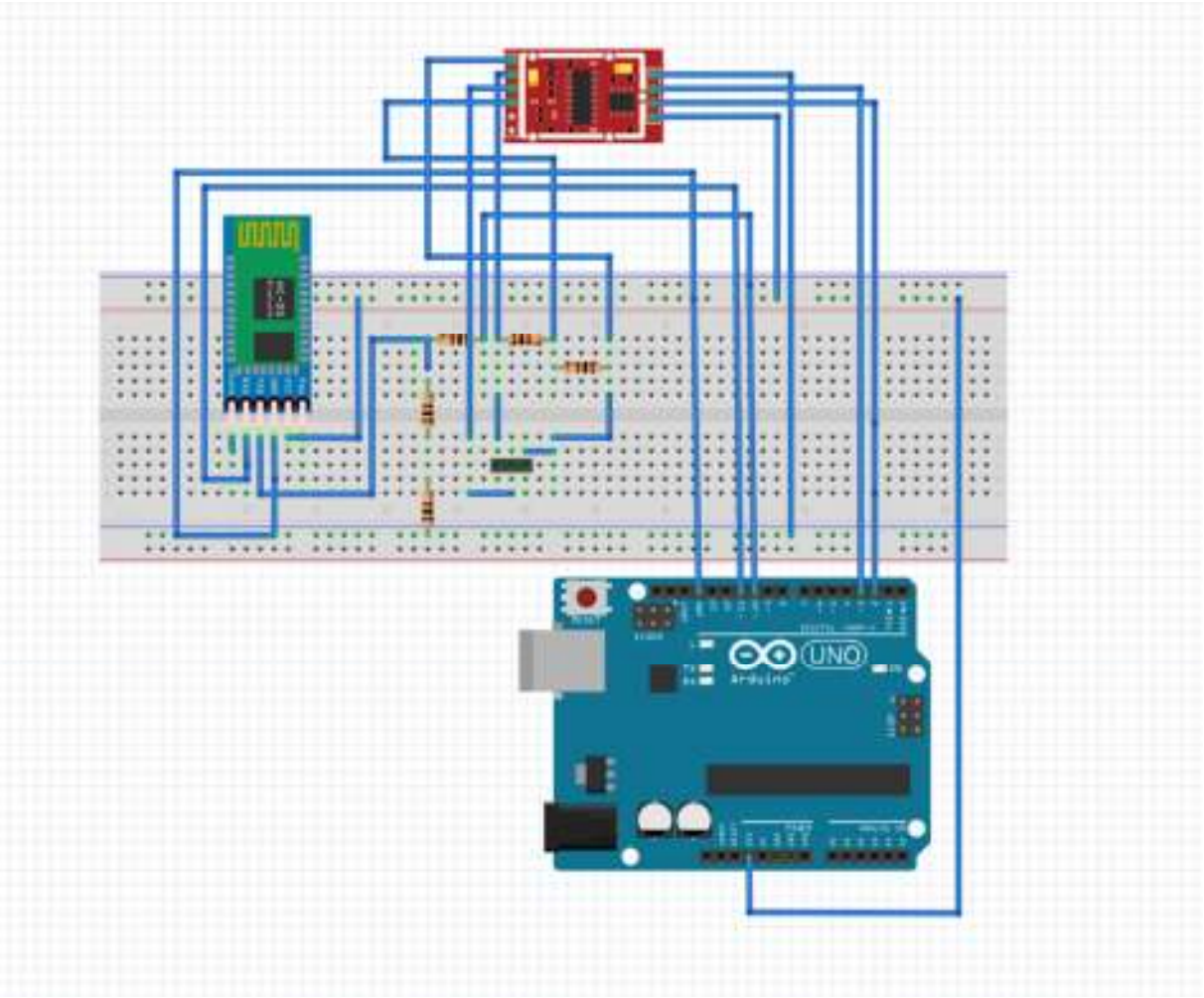
La descripción de las seis patillas del módulo es la siguiente:

a) RXD: es la entrada de recepción de datos al módulo, que debe conectarse a una salida del Arduino. Su nivel alto es de 3.3V. Dado que las salidas del Arduino trabajan a 5 V, no se pueden conectar directamente, so riesgo de dañar el módulo. Deben conectarse mediante un divisor de tensión (ver más adelante)
b) TXD: es la salida de transmisión de datos del módulo, que debe conectarse a una entrada del Arduino. También trabaja a 3.3 V, pero el Arduino lo reconoce como nivel alto, por lo que se puede conectar directamente.
c) VCC: entrada de alimentación, se puede conectar a las salidas de 5V o de 3.3 V del Arduino.
d) GND: se conecta a la salida de masa del Arduino.
e) STATE: salida que se pone a nivel alto cuando el módulo está activo.
f) EN: entrada que cuando está en alto, hace que el módulo entre en estado de comando, pudiendo ser programado por comandos AT[c] a través de RXD. Sin embargo, normalmente no es necesaria su utilización y se conecta a GND, o se deja sin conectar.

[c] Conjunto de instrucciones estandarizadas para controlar y comunicarse con dispositivos de comunicación remota. "AT" significa atención.

Se ha realizado un esquema en Fritzing que se puede encontrar también en el repositorio de Github antes mencionado[12]. En la biblioteca estándar no existía el módulo AT-09, por lo que se ha empleado el HC-05, que tiene idéntico patillaje.





Una vez hechas las conexiones físicas, hay que configurar el módulo, lo que hacemos ejecutando en el Arduino un código[13] que envía una lista de comandos AT para realizar dicha configuración. Este código sólo debe ser ejecutado una vez, ya que el módulo guarda estos comandos en memoria no volátil.

El siguiente paso que hicimos fue desarrollar un código que comunicara el módulo con un móvil [14]. Para ello, descargamos previamente en el móvil la aplicación *"Serial Bluetooth Terminal"*. El código envía la información del valor del puente de Wheatstone a la aplicación móvil, y a su vez puede recibir desde éste un comando (no es un comando estándar AT) para efectuar la tara.

2.3. Conexión con la Raspberry

Llegados a este punto diseñamos dos códigos:

a. Código en C++ [15] corriendo en Arduino que llevara el valor del peso a un dispositivo conectado por BLE
b. Código en Python [16], corriendo en la Raspberry que recibiera el valor del peso, y que le enviara un comando para hacer la tara, si fuera preciso.

3. Cálculo de la potencia

Finalmente, llegó el momento de realizar un código en Python[17] que recogiera los datos de la velocidad angular y de la fuerza de pedalada, y calculara la potencia mecánica desarrollada por el ciclista, reflejando en la pantalla:

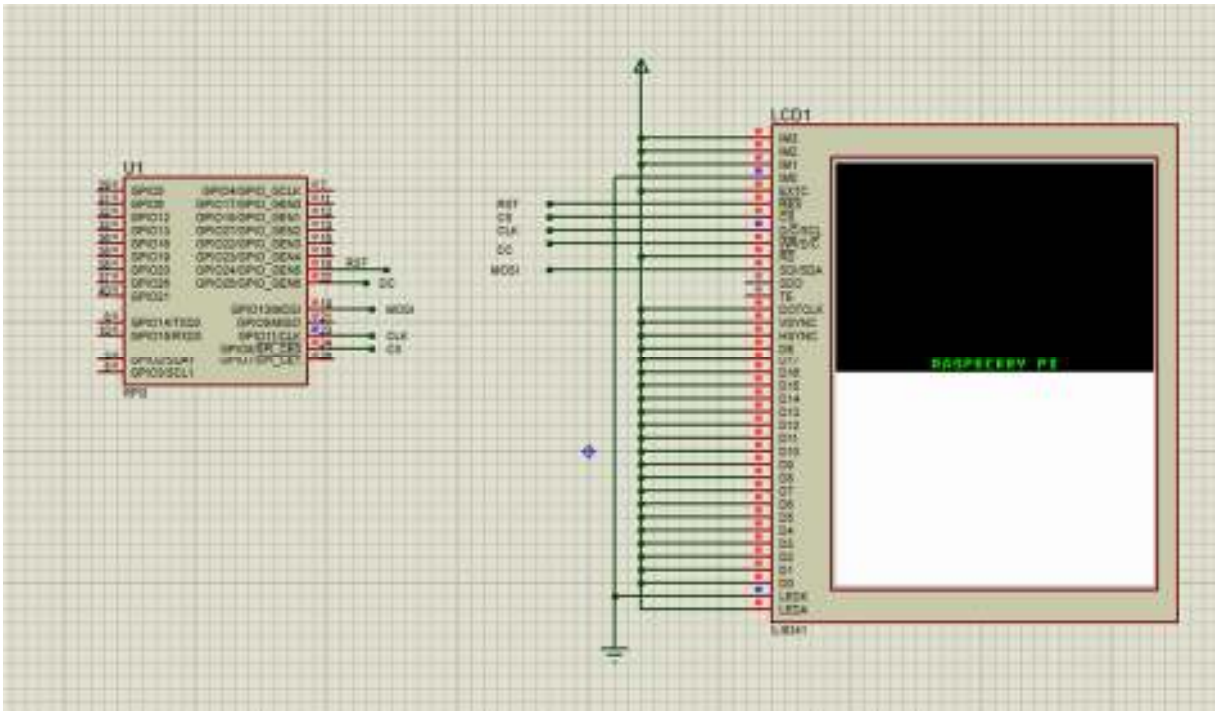
a. Velocidad angular de pedaleo en rpm
b. Velocidad lineal en Km/h
c. Distancia recorrida acumulada en Km
d. Peso en Kg aplicado al pedal
e. Fuerza en N aplicada al pedal
f. Par desarrollado en la biela en Nm
g. Potencia en vatios
h. Pulsos totales (giros de rueda) detectados

También crea un fichero de bitácora[18] que registra cada 5 segundos los datos mencionados arriba.

4. Visualización

Para la visualización de los datos en una consola, emplearemos una pantalla táctil TFT basada en el chip ILI9341, con entrada de datos serie.

Simulamos en Proteus esta pantalla, utilizando el modelo del ILI9341 que tiene disponible. Este modelo tiene además de la entrada de datos serie, 17 entradas de datos paralelo, que no utilizaremos.



La Raspberry envía los datos a la TFT vía el protocolo de comunicación serie SPI (*"Serial Peripheral Interface")*, utilizando las salidas dedicadas MOSI (*"Master Out Slave In")* y CLK (*"clock")*, que deben conectarse a las entradas SD y SCK de la TFT.

Además, la TFT tiene que recibir 3 entradas:

a. RST: hace un *reset* cuando está a nivel bajo. En funcionamiento normal siempre tiene que estar en alto. Puede venir de cualquier GPIO de la Raspberry.
b. CS: *"chip selector"*. Normalmente está en nivel alto. Tiene que estar en nivel bajo para que se active la interface SPI de la TFT, y volver a nivel alto para que cese dicha activación. Normalmente se emplea el SPI CEO (pin 24) de la Raspberry, aunque puede emplearse otro GPIO general.
c. DC: selector entre datos (*HIGH*) y comandos (*LOW*). Puede emplearse cualquier GPIO

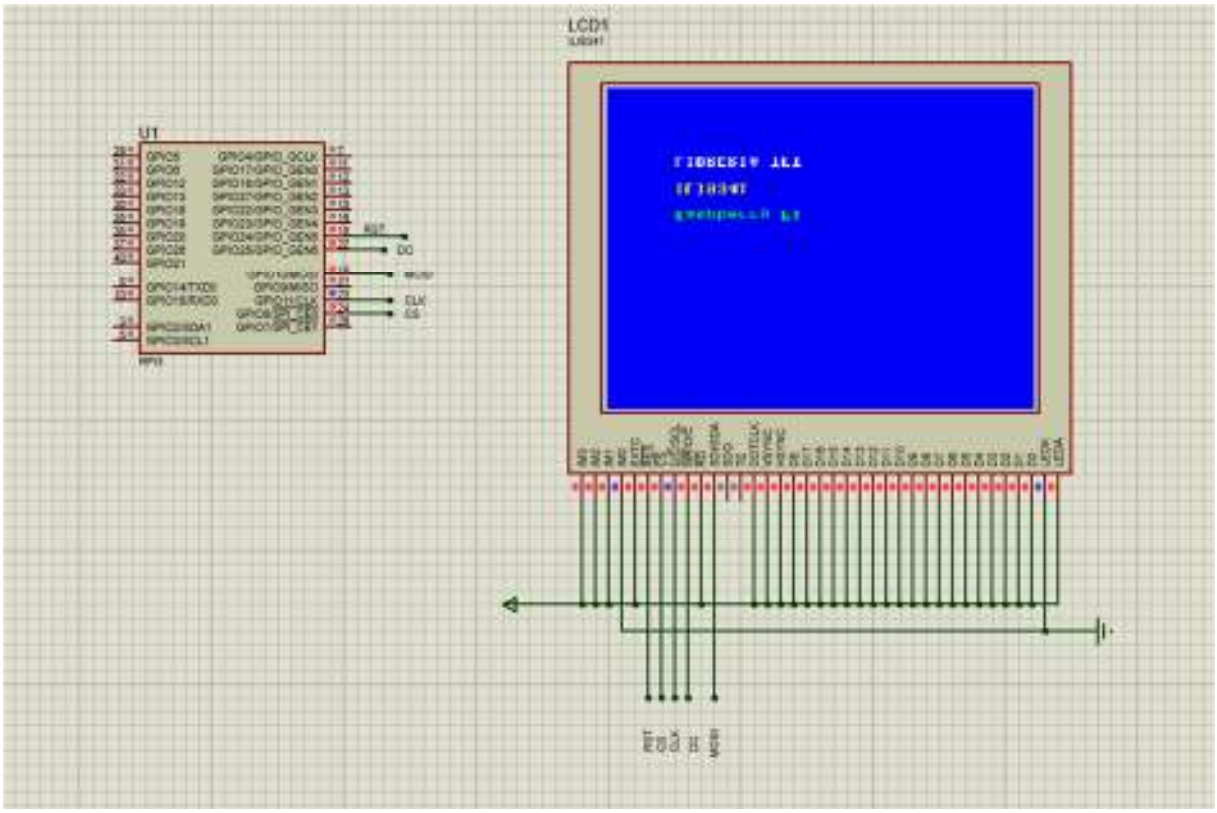
Simulamos un programa sencillo de prueba con resultado satisfactorio, por lo que procedimos a probarlo en la Raspberry. Para ello hay que verificar dos puntos en la misma:

a) que tenga instalada la librería *SPIDEV*
b) que tenga habilitada la SPI (ver en las opciones de Interface de la configuración)

Una vez verificados estos puntos, procedimos a ejecutar el código, con resultados satisfactorios.

Con vistas a utilizar la TFT en los diversos códigos procedimos a realizar un fichero de librería[19], y un documento en PDF con la descripción de las funciones y parámetros [20]

Entre las funciones desarrolladas están las que permite rotar el texto en la pantalla (funciones *" init ()"* y *"set_rotacion()"*). Hicimos un código de prueba para escribir en una pantalla apaisada (es decir girada 90º)[21]. Al simularlo en Proteus, obtuvimos curiosamente que los textos se escribían de forma especular:



No obstante, al ejecutar el código en la Raspberry encontramos que las letras se escribían correctamente.

5. Conclusiones

Se ha verificado la posibilidad de realizar un medidor de potencia mecánica de bajo coste. Obviamente, por problemas de espacio en el pedal, debe sustituirse el Arduino Uno, por un microcontrolador más pequeño, tipo Arduino nano, Raspberry Pi Pico o ATtiny85, así como implementar un sistema de alimentación autónoma para esta parte.

Referencias

[1] Hall.py

[2] https://raspberrypi.stackexchange.com/questions/147332/rpi-gpio-runtimeerror-failed-to-add-edge-detection

[3] Línea 85 del fichero Hall.py

[4] Hall_log.csv

[5] Línea 117 del fichero Hall.py

[6] Celula_carga.py

[7] https://github.com/tatobari/hx711py

[8] Sensor_carga.py

[9] https://github.com/bogde/HX711

[10] Celula_carga.h

[11] https://www.thingiverse.com/thing:4895558

[12] BLE. fzz

[13] Configuracion_BLE.h

[14] Celula_carga_BLE.h

[15] Sensor_carga_rasp.h

[16] Celula_carga_BLE.py

[17] Medidor de potencia.py

[18] sistema_completo_log.csv

[19] TFT_ILI9341.py

[20] Libreria_TFT.pdf

[21] test_tft.py