UNIVERSIDAD CATÓLICA DE CÓRDOBA
Universidad Jesuita

ucc | FACULTAD DE INGENIERÍA

Ingeniería Electrónica

Proyecto y Diseño

Profesores: Ducloux, José María; Germena, Daniel

Alumno: Nievas Aramayo, Pablo Agustín
30/01/2022

i. Índice

Resumen

En el presente informe se desarrolla la implementación de un velocímetro y un cronometro con *Displays Led* 7 segmentos sobre la plataforma *Rapberry Pi*, programado con el lenguaje *Python,* como parte de un proyecto del departamento de investigación de ingeniería mecánica de la Universidad Católica de Córdoba. Para mantener una separación entre lo mecánico y lo electrónico se emplea un sensor *Hall*.

La mayoría de los requerimientos y decisiones sobre este proyecto se tomaron en agosto de 2019, pensado sobre un equipo mecánico de rehabilitación y/o competencia de un usuario de silla de ruedas, montado sobre unos rodillos, incluso con posibles infantes en la cercanía.

Palabras Claves

Raspberry Pi, Display, Sensor *Hall*, *Python*, Silla de ruedas, rodillos.

ii. Índice de Figuras

iii. Índice de Tablas

1. Introducción

La profesora de educación física de la "Escuela Especial Beatriz Angélica Martínez de Allio" presenta los requisitos funcionales y no funcionales, visible en la Tabla 1.1.

Tabla 1.1 Requisitos Funcionales y No funcionales

Requisitos Funcionales	Requisitos No Funcionales
Cronometrar el tiempo de uso	Económico/Barato
Calcular la velocidad lineal de la silla de Ruedas	Robusto debido a la presencia de infantes
Mostrar visualmente, con luz, los previos requisitos	Pocos elementos
	Fácil adquisición
	Libre acceso al *software y hardware*

Se plantea como límites para los requerimientos no funcionales los presentes en la Tabla 1.2:

Tabla 1.2 Especificaciones de requerimientos no funcionales

Económico/Barato	Costo del sistema menor a 12 (doce) mil pesos
Robusto debido a la presencia de infantes	Imposibilidad física de infantes de contacto directo con el interior del sistema
Pocos elementos	Límite de 20 elementos que componen al sistema
Fácil adquisición	Posibilidad de compra local de todos los elementos del sistemas
Libre acceso al *software y hardware*	Tanto los elementos del sistema como la documentación del proyecto son de libre acceso

Se plantea como límite de los requerimientos funcionales los presentados en la Tabla 1.3:

Tabla 1.3 Especificaciones de requerimientos funcionales

Cronometrar el tiempo de uso	Mostrar en segundos el tiempo real transcurrido al usar el sistema, como mínimo 60 segundos
Calcular la velocidad lineal de la silla de Ruedas	Velocidad de la silla en kilómetros por hora
Mostrar visualmente, con luz, los previos requisitos	Los números que muestra el sistema deben de ser visibles a más de 2 metros de distancia del sistema como también visibles con poca luz

2. Marco Teórico

El proyecto se presenta en 3 tipos de elementos que merecen una explicación teórica: Subcircuitos electrónicos, *Raspberry Pi* y Código.

Subcircuitos electrónicos:

Pull up y *Pull down* es un circuito electrónico para mantener un nivel de tensión continua en una entrada para no dejarla al “aire”.

Una entrada al “aire” se refiere a aquella entrada digital que no está conectada directamente a un nivel de tensión, incluyendo masa (0 Voltios [V]). Una entrada al aire es susceptible al ruido, funcionando como una pequeña antena, con la posibilidad de variar su nivel de tensión de la entrada entre apagado y prendido.

En caso de que la señal digital que deseamos visualizar es de una “llave” en alto, entonces se conecta a la entrada un *Pull down,* ilustrado en la Figura 2.1.

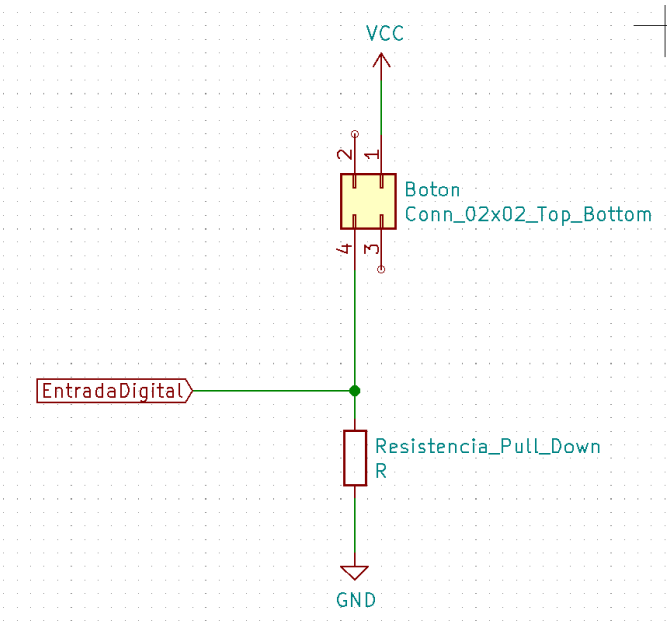


Figura 2.1 Conexión de entrada con *Pull Down*

Si tenemos un elemento externo que su efecto es hacer un corto entre un pin y masa, para conectarlo sin que quede la entrada digital al aire, se conecta con un *Pull up* como se ve en la Figura 2.2:

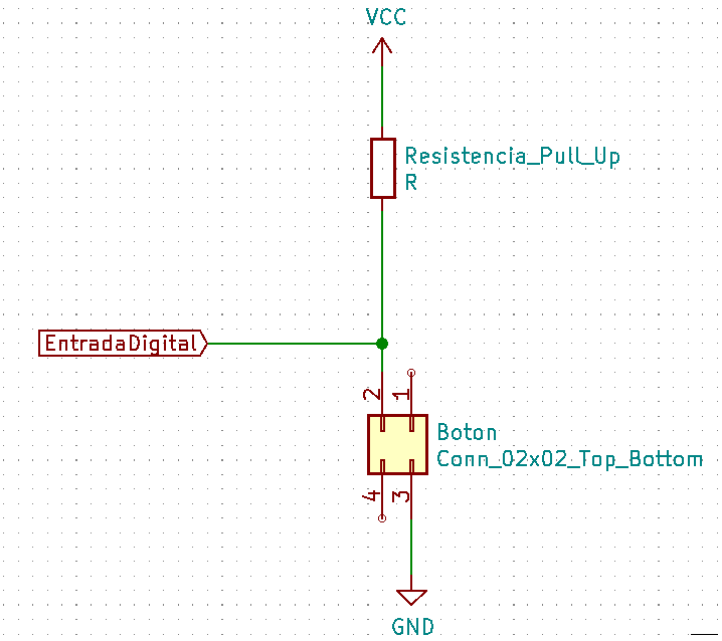


Figura 2.2 Conexión de entrada con *Pull Up*

En la Figura 2.3 [4] se observa un *Display Led* 7 segmentos junto con su circuito de diodos equivalente, cada segmento es un *led* individual. Este *display* es de cátodo común, por lo que todos los cátodos están conectados al pin de masa.

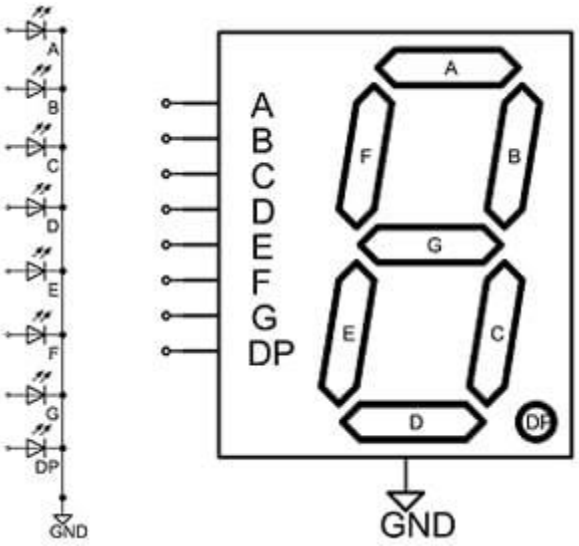


Figura 2.3 *Display Led* 7 segmentos [4]

Método de multiplexado: Consiste en intercambiar la conexión del cátodo o ánodo común de cada *display* individual, por ejemplo con el uso de transistores, de los pines de cátodo común a masa. Siguiendo la ecuación de "n" igual a la cantidad de cables y de resistencias, y "D" la cantidad de *displays* en las siguientes expresiones (1), (2):

Ecuación sin multiplexado:

$$n = 7 * D$$

(1)

Ecuación multiplexado:

$$n = 7 + D$$

(2)

De tal manera que se simplifica la conexión de los *displays* para poder utilizar una cantidad "m" menos de cables y de resistencias (3).

$$m = (7 * D) - 7 + D$$

(3)

En los *Displays Led* 7 segmentos requieren de conectar en serie a cada diodo una resistencia.

La polarización de un transistor NPN de emisor común, visualizado en la Figura 2.4, es gobernada por las ecuaciones (4) y (5)

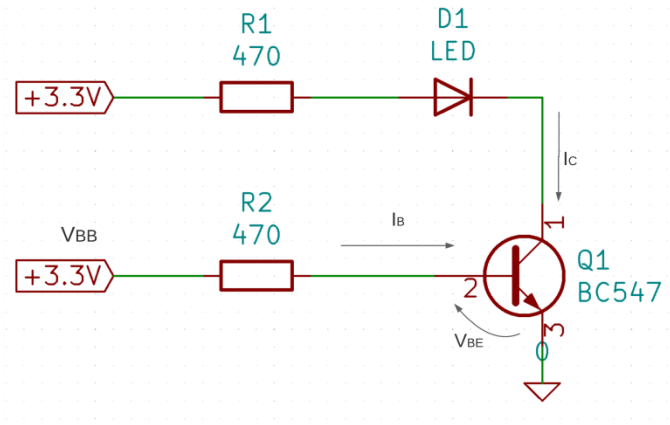


Figura 2.4 Polarización de transistor NPN BJT

Relación de corriente entre colector y base:

$$I_c = \beta * I_B$$

(4)

β: Factor de ganancia de corriente

I_B: Corriente de base

I_C: Corriente de colector

Corriente de base dependiente de la tensión y resistencia de entrada:

$$I_B = \frac{V_{BB} - V_{BE}}{R2}$$

(5)

V_{BB}: Tensión de alimentación de base

V_{BE}: Tensión entre base y emisor de transistor

El sensor *hall* provee una señal en respuesta a la presencia de un campo magnético, analógica o digital. En la Figura 2.5 se presenta la salida digital típica y la señal V_o con su histéresis [5].

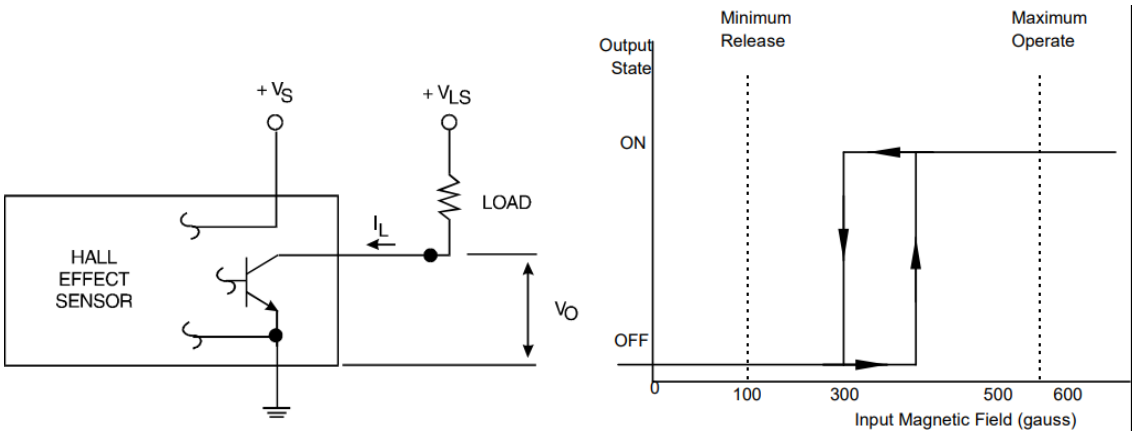


Figura 2.5 Sensor *hall* y señal Vo [5]

Raspberry Pi:

Las *Raspberry pi* son computadoras del esquemático de la Figura 2.6 [14], mientras que la compatibilidad de los *pines* GPIO se mantiene entre distintas versiones, no así la forma de la placa, ilustrada en la Figura 2.7 [7].

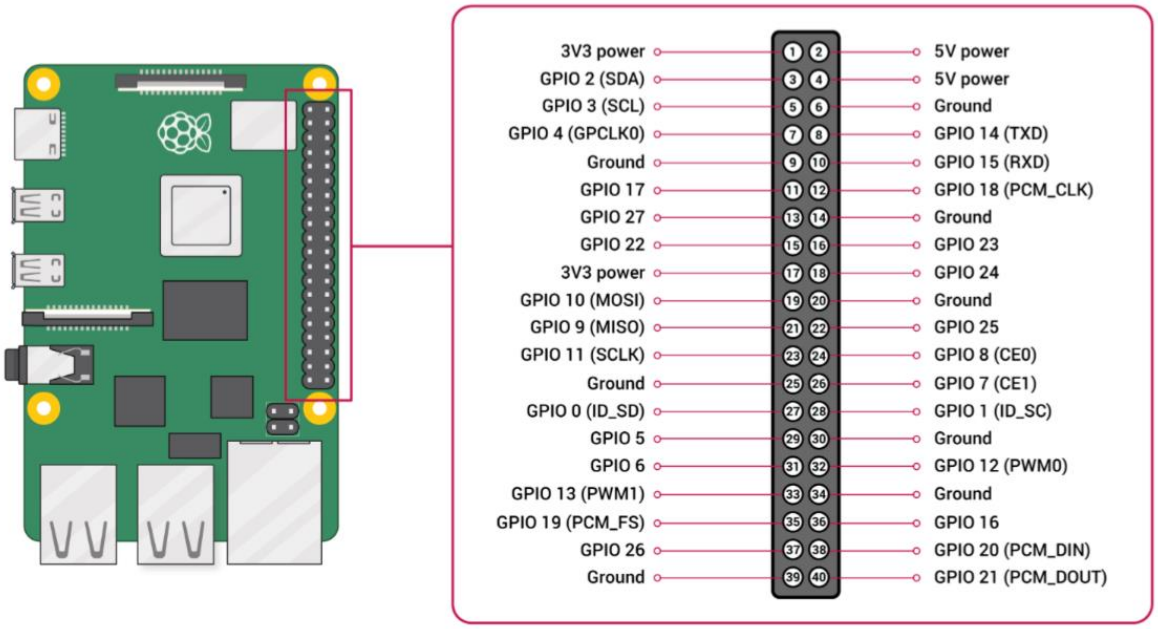


Figura 2.6 Esquemático simple de placa Raspberry Pi

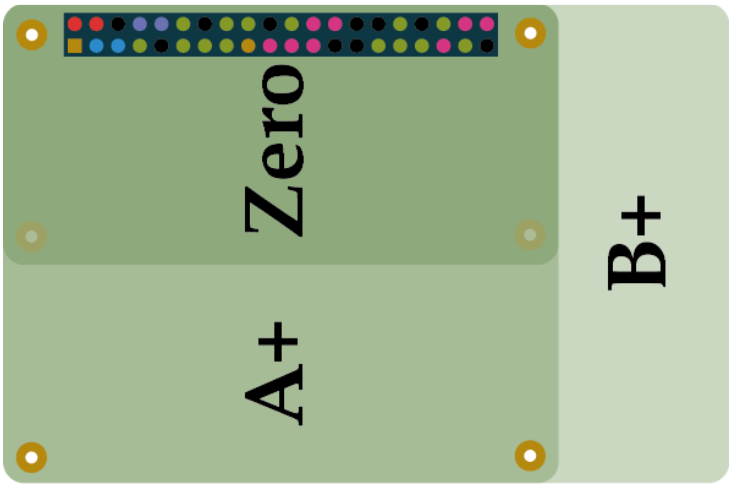


Figura 2.7 GPIO de distintas versiones de la familia *Raspberry Pi*

A las *Raspberry Pi* se le instala un sistema operativo en la tarjeta SD, siguiendo las instrucciones en el sitio web oficial de *Raspberry Pi* [14].

Comandos de control y navegación se presentan en la Tabla 2.1.

Tabla 2.1 Comandos de control en SO base Linux

cd	*Change directory*, se utiliza para cambiar el directorio sobre el cual se está trabajando
mkdir	Crea un directorio, carpeta
sudo	*Super User DO,* ejecuta el comando adjunto con privilegio elevado, como “administrador”
nano	Abre un editor de texto de archivo
python	Ejecuta el archivo con el intérprete de *Python*
ls	Enlista los archivos contenidos en un directorio
chmod 755	*Change Mode,* cambia el modo de acceso de un archivo.755 perite la ejecución del archivo

sh	*Bourne Shell,* hace compatible un código en un archivo con el sistema operativo
reboot	Resetea el sistema
crontab -e	Abre una ventana *crontab* para ejecutar comandos con una agenda específica.

Código:

Funciones de lenguaje *Python* de control sobre una *Raspberry Pi* que conciernen al proyecto [9] están expresadas en la Tabla 2.2:

Tabla 2.2 Funciones de programación en *Python Raspberry Pi*

import RPi.GPIO as GPIO	Importa el módulo RPi.GPIO como GPIO
GPIO.setmode(GPIO.BCM)	Enumera las GPIO según el número de canal en el *Broadcom SOC*
GPIO.setwarnings(False)	Deshabilita las alarmas cuando un pin se configura a algo distinto a lo *default*
GPIO.setup(channel, GPIO.IN)	Configura un canal como un pin de entrada
GPIO.setup(channel, GPIO.OUT)	Configura un canal como un pin de salida
chan_list = [11,12]	Lista con números de posibles canales
GPIO.setup(chan_list, GPIO.OUT)	Configura varios canales al mismo tiempo como salida
GPIO.input(channel)	Retorna el valor de entrada del canal.
GPIO.output(channel, state)	Configura el estado del canal especifico
GPIO.output(chan_list, GPIO.LOW)	Configura el estado de los canales enumerados en la lista

GPIO.output(chan_list, (GPIO.HIGH, GPIO.LOW))	Configura individualmente cada estado de los canales de la lista
GPIO.setup(channel, GPIO.IN, pull_up_down=GPIO.PUD_UP)	Conecta una resistencia de 10Kohm entre el canal de entrada y 3.3V, aparte de configurar el canal como in pin de entrada
GPIO.setup(channel, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)	Conecta una resistencia de 10Kohm entre el canal de entrada y 0V, aparte de configurar el canal como in pin de salida
GPIO.add_event_detect(channel, GPIO.RISING)	Agrega un evento que detecta un flanco ascendente en el canal.
GPIO.event_detected(channel)	Retorna un *True* al detectar el evento.
def shut_down()	Función que al ser llamada apaga el sistema

Para terminar el capítulo 2 Marco Teórico, se presenta el cálculo de velocidad de la silla de ruedas con las ecuaciones:

$$V = \omega * r$$

$$\omega = \frac{2 * \pi}{T}$$

$$V = \frac{2 * \pi * r}{T}$$

V: Velocidad lineal

ω: Velocidad angular

r: radio de circulo

T: Tiempo de círculo en dar una vuelta completa

Al estar los rodillos en contacto con las sillas de ruedas, ambos rotan a distinta velocidad angular, pero a la misma velocidad lineal. Por lo que al calcular la velocidad lineal del rodillo se obtiene la de la silla de ruedas.

3. Diseño

Partiendo de lo presente en el capítulo 1, se diseña el diagrama en bloques de la Figura 3.1, que consta de 5 bloques. Dos bloques en los que donde se muestra la información, siendo una posibilidad la “Nube” pero una prioritaria visualización física en “Visualización”. Un bloque de “Procesamiento”, donde se trabaja sobre todas las señales del sistema y computa la matemática y la lógica para la “Visualización”.

Se tiene un bloque de “Control” que permite al usuario interactuar con el sistema para distintos funciones de la aplicación presente en el bloque de “Procesamiento”. Y por último se tiene el bloque de los “Sensores”, que enviará las señales a procesar.

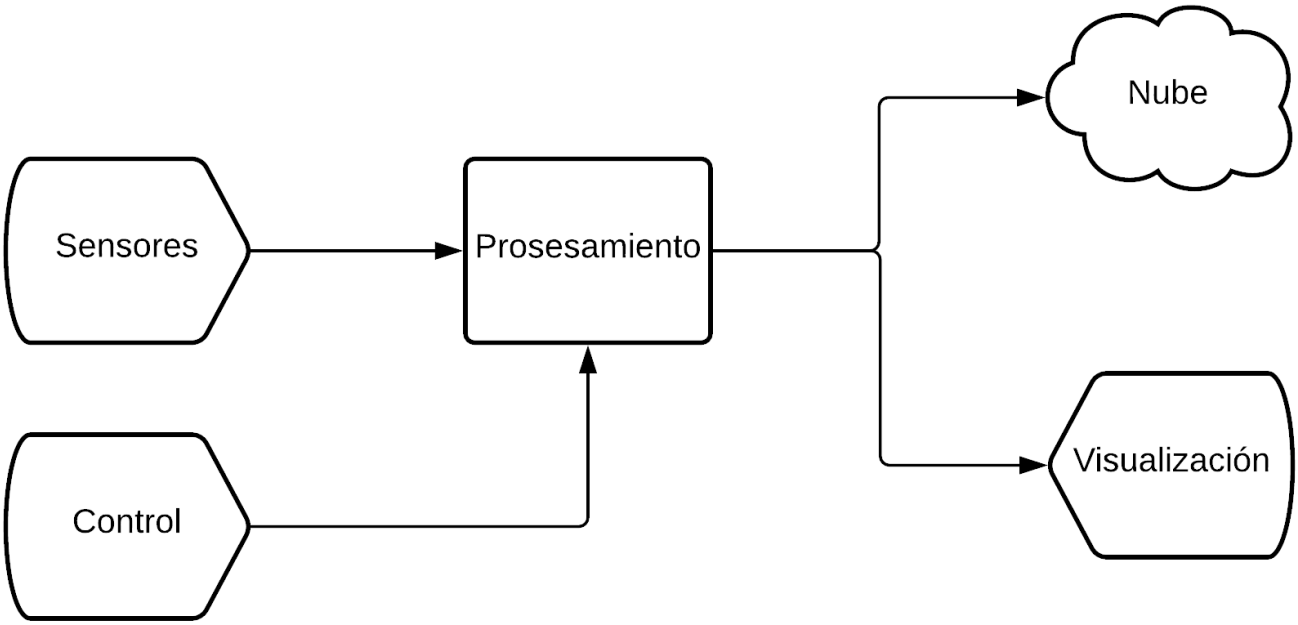


Figura 3.1 Diagrama en bloques de Proyecto

Se obtiene la Figura 3.2 al colocar el diagrama en bloques en contexto con el sistema físico sobre el cual se monta el proyecto.

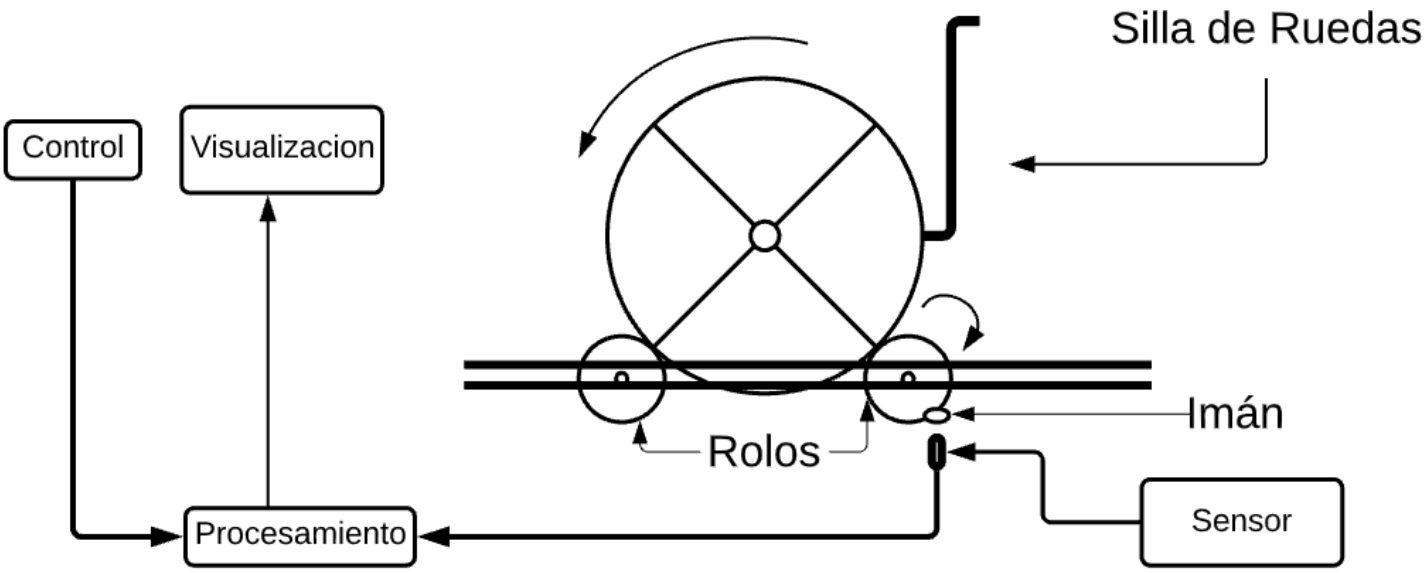


Figura 3.2 Bloques del sistema contextualizados

El procesamiento a realizar corresponde al diagrama de flujo de la Figura 3.3.

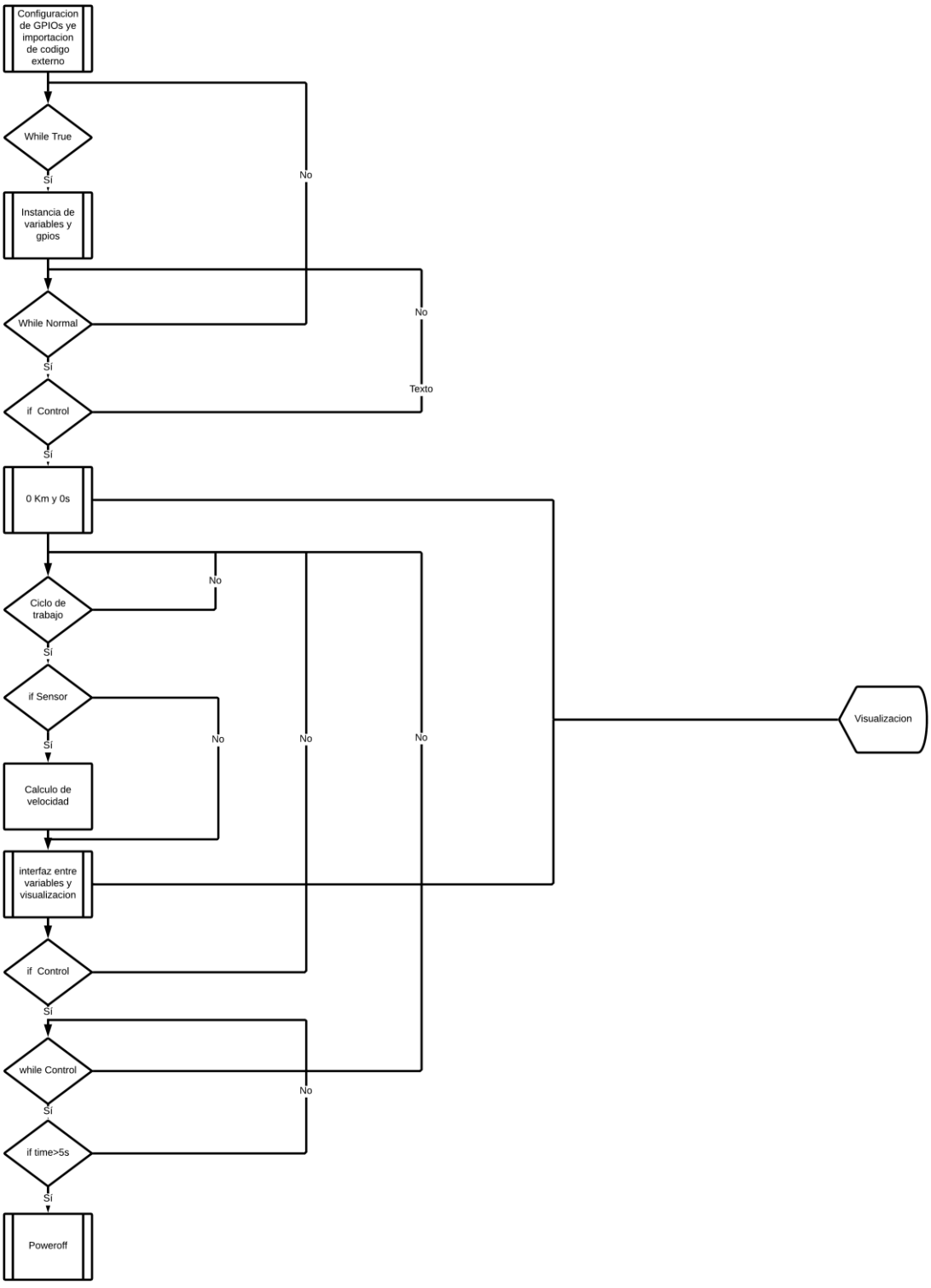


Figura 3.3 Diagrama de flujo de procesamiento

4. Implementación

En el bloque de “Visualización” se descarta el uso de una pantalla *LCD* por su baja legibilidad con poca luz y su costo, por lo que se opta por *displays led* 7 segmentos. La creación de un sistema preparado para estos *displays* permite un fácil cambio de implementación al escalar la potencia y tamaño de los *displays* por medio del control de multiplexado.

Se utilizan 4 *displays led* 7 segmentos SC56-11SR [2] en total, 2 para el tiempo como un cronómetro, y otros 2 para mostrar los kilómetros por hora. Al contar con 2 *displays*, se fija la precisión de medición de velocidad en 0,5 Km/h; siendo la resolución de 1 Km/h. De igual manera, el cronometro tiene un rango de 99 segundos decimales.

El control por multiplexado se produce con el uso de transistores NPN BJT BC546 [3].

Se implementa el uso de un solo pulsador para el bloque de “Control” en favor de la simpleza. El usuario arranca el funcionamiento normal del sistema al accionar el pulsador, si este esta energizado. En ese mismo momento se inicia tanto el cronometro como el velocímetro. Si el usuario desea reiniciar los marcadores debe presionar el pulsador una vez más. Para quitar la energía en forma segura del sistema, primero debe apagarlo manteniendo apretado el pulsador durante 5 segundos.

Ante la opción de usar un sensor infrarrojo o un sensor *hall*, se utiliza este último ya que es visible y de manera práctica su funcionamiento mediante el *led* de la placa que presenta el módulo Arduino sensor *hall* 3144 [1]. Además, es robusto ante factores externos, no requiere contacto físico con el rodillo de manera directa y no depende de la luminosidad o colores del sistema.

Se opta por utilizar como plataforma de procesamiento la serie de computadoras *Raspberry Pi* sobre otras por la posibilidad de escalar el sistema a varios usuarios, mantenimiento técnico, y fácil adquisición en el mercado

local [16]. El proyecto de desarrolla sobre la *Raspberry Pi* 3B+. Documentación de esta computadora se encuentra en la respectiva página oficial [14].

Para utilizar la *Raspberry Pi* es necesario conectarla a la red eléctrica, esto se realiza con el uso de un cargador de celular [17].

Los elementos del sistema se ven conectados en la Figura 4.1:

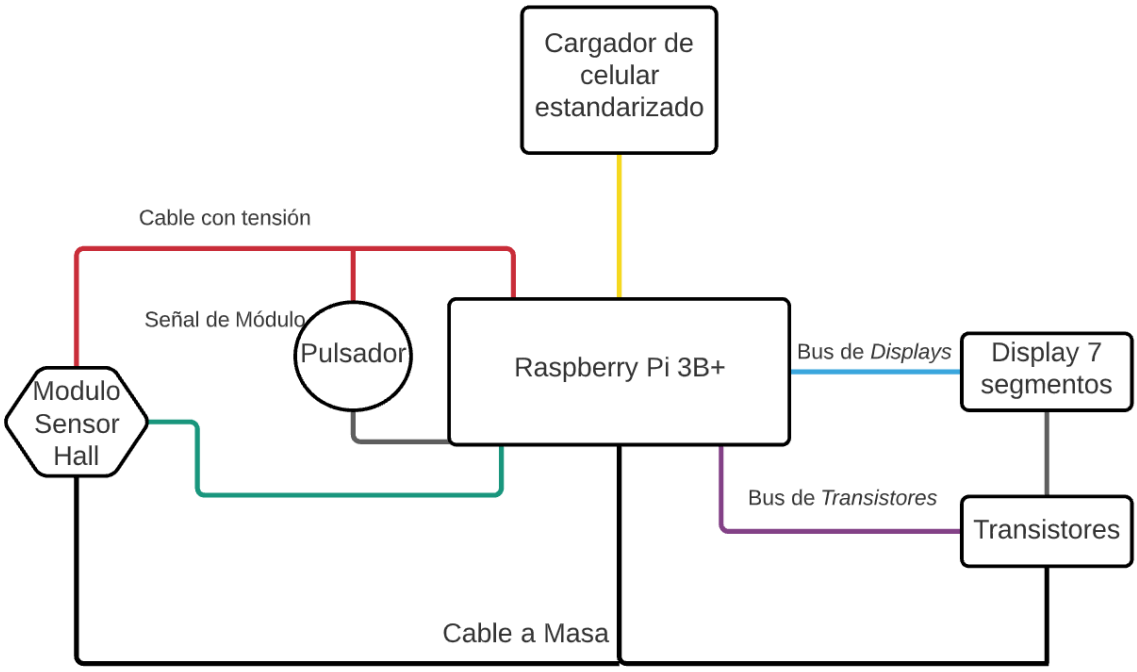


Figura 4.1 Implementación de interconexión del sistema

A la *Raspberry Pi* se le instala un sistema operativo, *Raspberry Pi OS Lite*, en la tarjeta SD. Esto se realiza con el programa *Raspberry Pi Imager*, en una computadora con *Windows,* siguiendo las instrucciones de la misma página de *Raspberry Pi* [4].

Con el inicio de sesión por *default*: pi, Raspberry Pi. Se ingresan los comandos “cd ..” hasta llegar al directorio raíz. En ese punto, se crea una carpeta llamada “hallsensor” con el comando: “mkdir hallsensor”.

En la carpeta “hallsensor” se crea el código en *Python* llamado hallsensor.py con el comando: “sudo nano hallsensor.py”.

El código de *Python* del diagrama de flujo, Figura 3.3, es:

```
def shut_down():
    print("shutting down")
    command = "/usr/bin/sudo /sbin/shutdown -h now"
    import subprocess
    process = subprocess.Popen(command.split(), stdout=subprocess.PIPE)
    output = process.communicate()[0]
    print(output)
import RPi.GPIO as gpio
import time
gpio.setmode(gpio.BCM)
gpio.setwarnings(False)
while True:
        Funcionamiento_normal=1
        Tiempo_para_apagar=0
        Tiempo_inicio=0
        Tiempo_trabajo=0
        Tiempo_diferencia=1
        Tiempo_diferencia_inicial=0
        Tiempo_diferencia_final=1
        Segundos_unidad=0
        Segundos_decena=0
        Segmento_display_led_a=23
        Segmento_display_led_b=24
        Segmento_display_led_c=8
        Segmento_display_led_d=25
        Segmento_display_led_e=14
        Segmento_display_led_f=15
        Segmento_display_led_g=18
        Segmentos_display_todos=[a,b,c,d,e,f,g]
        Velocidad_total=0
```

```
Velocidad_unidad=8

Velocidad_decena=8

Display_velocidad_unidad=1

Display_velocidad_decena=7

Display_segundos_unidad=12

Display_segundos_decena=16

Radio_rodillo=0.1

Sensor_hall=20

Pulsador_control=21

gpio.setup(Sensor_hall, gpio.IN)

gpio.setup(Pulsador_control, gpio.IN, pull_up_down=gpio.PUD_DOWN)

gpio.setup(Segmentos_display_todos, gpio.OUT)

gpio.setup(Display_velocidad_unidad, gpio.OUT)

gpio.setup(Display_velocidad_decena, gpio.OUT)

gpio.setup(Display_segundos_unidad, gpio.OUT)

gpio.setup(Display_segundos_decena, gpio.OUT)

gpio.add_event_detect(Sensor_hall, gpio.FALLING)

gpio.add_event_detect(Pulsador_control, gpio.RISING)

gpio.output(Display_velocidad_unidad, gpio.LOW)

gpio.output(Display_velocidad_decena, gpio.LOW)

gpio.output(Display_segundos_unidad, gpio.LOW)

gpio.output(Display_segundos_decena, gpio.LOW)

gpio.output(Segmentos_display_todos, gpio.LOW)

Numero_segmentos_display={

                0: (gpio.HIGH, gpio.HIGH, gpio.HIGH, gpio.HIGH, gpio.HIGH, gpio.HIGH, gpio.LOW)

                1: (gpio.LOW, gpio.HIGH, gpio.HIGH, gpio.LOW, gpio.LOW, gpio.LOW, gpio.LOW)

                2: (gpio.HIGH, gpio.HIGH, gpio.LOW, gpio.HIGH, gpio.HIGH, gpio.LOW, gpio.HIGH)

                3: (gpio.HIGH, gpio.HIGH, gpio.HIGH, gpio.HIGH, gpio.LOW, gpio.LOW, gpio.HIGH)

                4: (gpio.LOW, gpio.HIGH, gpio.HIGH, gpio.LOW, gpio.LOW, gpio.HIGH, gpio.HIGH)

                5: (gpio.HIGH, gpio.LOW, gpio.HIGH, gpio.HIGH, gpio.LOW, gpio.HIGH, gpio.HIGH)

                6: (gpio.HIGH, gpio.LOW, gpio.HIGH, gpio.HIGH, gpio.HIGH, gpio.HIGH, gpio.HIGH)

                7: (gpio.HIGH, gpio.HIGH, gpio.HIGH, gpio.LOW, gpio.LOW, gpio.HIGH, gpio.LOW)
```

```
                    8: (gpio.HIGH, gpio.HIGH, gpio.HIGH, gpio.HIGH, gpio.HIGH, gpio.HIGH, gpio.HIGH)
                    9: (gpio.HIGH, gpio.HIGH, gpio.HIGH, gpio.LOW, gpio.LOW, gpio.HIGH, gpio.HIGH)
                    }
while Funcionamiento_normal:
        if (gpio.event_detected(Pulsador_control))
                    gpio.output(Display_velocidad_unidad, gpio.HIGH)
                    gpio.output(Segmentos_display_todos, Numero_segmentos_display[0])
                    time.sleep(0.4)
                    gpio.output(Display_velocidad_unidad, gpio.LOW)
                    gpio.output(Display_velocidad_decena, gpio.HIGH)
                    time.sleep(0.4)
                    gpio.output(Display_velocidad_decena, gpio.LOW)
                    gpio.output(Display_segundos_unidad, gpio.HIGH)
                    time.sleep(0.4)
                    gpio.output(Display_segundos_unidad, gpio.LOW)
                    gpio.output(Display_segundos_decena, gpio.HIGH)
                    time.sleep(0.4)
                    gpio.output(Display_segundos_decena, gpio.LOW)
                    Tiempo_inicio=time.time()
                    Tiempo_diferencia_inicial=time.time()
                    Run=1
                    while Run:
                            apagando=0
                            if(gpio.event_detected(Sensor_hall)):
                                    Tiempo_diferencia_final=time.time()
                                    Tiempo_diferencia=Tiempo_diferencia_final-Tiempo_diferencia_inicial
                                    Tiempo_diferencia_inicial=time.time()
                                    beta=(2*3.1415926535*Radio_rodillo)/Tiempo_diferencia
                                    Velocidad_total=beta*3.6/4
                                    Velocidad_total=int(round(Velocidad_total))
                                    Velocidad_unidad=int(Velocidad_total%10)
                                    Velocidad_decena=int((Velocidad_total-Velocidad_unidad)/10)
```

```
                Tiempo_trabajo=int(round(time.time() - Tiempo_inicio))
                if ((time.time()-Tiempo_diferencia_inicial)>20):
                        Velocidad_total=0
                Segundos_unidad=int(Tiempo_trabajo%10)
                Segundos_decena=int(((Tiempo_trabajo-Segundos_unidad)/10)%10)
                gpio.output(Display_velocidad_unidad, gpio.HIGH)
                gpio.output(Segmentos_display_todos, Numero_segmentos_display[Velocidad_unidad])
                time.sleep(0.005)
                gpio.output(Display_velocidad_unidad, gpio.LOW)
                gpio.output(Segmentos_display_todos, Numero_segmentos_display[Velocidad_decena])
                gpio.output(Display_velocidad_decena, gpio.HIGH)
                time.sleep(0.005)
                gpio.output(Display_velocidad_decena, gpio.LOW)
                gpio.output(Segmentos_display_todos, Numero_segmentos_display[Segundos_unidad])
                gpio.output(Display_segundos_unidad, gpio.HIGH)
                time.sleep(0.005)
                gpio.output(Display_segundos_unidad, gpio.LOW)
                gpio.output(Segmentos_display_todos, Numero_segmentos_display[Segundos_decena])
                gpio.output(Display_segundos_decena, gpio.HIGH)
                time.sleep(0.005)
                gpio.output(Display_segundos_decena, gpio.LOW)
                if gpio.input(Pulsador_control)==True:
                        Run=0
                        while gpio.input(Pulsador_control)== True:
                                Tiempo_para_apagar=Tiempo_para_apagar+0.05
                                if Tiempo_para_apagar > 1000:
                                        shut_down()
gpio.output(Display_velocidad_unidad, gpio.HIGH)
gpio.output(Segmentos_display_todos, Numero_segmentos_display[Velocidad_unidad])
time.sleep(0.005)
gpio.output(Display_velocidad_unidad, gpio.LOW)
gpio.output(Segmentos_display_todos, Numero_segmentos_display[Velocidad_decena])
```

```
gpio.output(Display_velocidad_decena, gpio.HIGH)

time.sleep(0.005)

gpio.output(Display_velocidad_decena, gpio.LOW)

gpio.output(Display_segundos_unidad, gpio.HIGH)

gpio.output(Segmentos_display_todos, Numero_segmentos_display[Segundos_unidad])

time.sleep(0.005)

gpio.output(Display_segundos_unidad, gpio.LOW)

gpio.output(Segmentos_display_todos, Numero_segmentos_display[Segundos_decena])

gpio.output(Display_segundos_decena, gpio.HIGH)

time.sleep(0.005)

gpio.output(Display_segundos_decena, gpio.LOW)
```

Se crea un archivo ejecutable tipo *shell* para que el código *Python* sea ejecutable al prender la computadora *Raspberry Pi*, usando el comando nano. El código de la *shell*, llamada “launcher.sh”, se ve en la Figura 4.3

```
1   cd /
2   cd hallsensor
3   sudo python hallsensor.py
4   cd /
```

Figura 4.3 Código *shell*

Al tener el launcher.sh creado, se ejecuta el comando “chmod 755 launcher.sh”.

Se crea una carpeta, utilizando el comando “mkdr”, para guardar los errores a la hora de ejecutar el código; esta carpeta es llamada “*logs*”.

Se ejecuta el comando “sudo crontab –e” para agregar el comando en la ventana emergente la línea [8]:

“ @reboot sh /home/pi/bbt/launcher.sh >/home/pi/logs/cronlog 2>&1”

En el apartado del *hardware*, se emplea una placa de cobre para las pruebas y medición de las señales del sistema. El esquemático de la placa, Figura 4.4, junto con el diseño del *PCB*, Figura 4.5, con la herramienta KiCad [13] para las pruebas de las señales.

Figura 4.4 Esquemático de la electrónica del sistema en KiCad

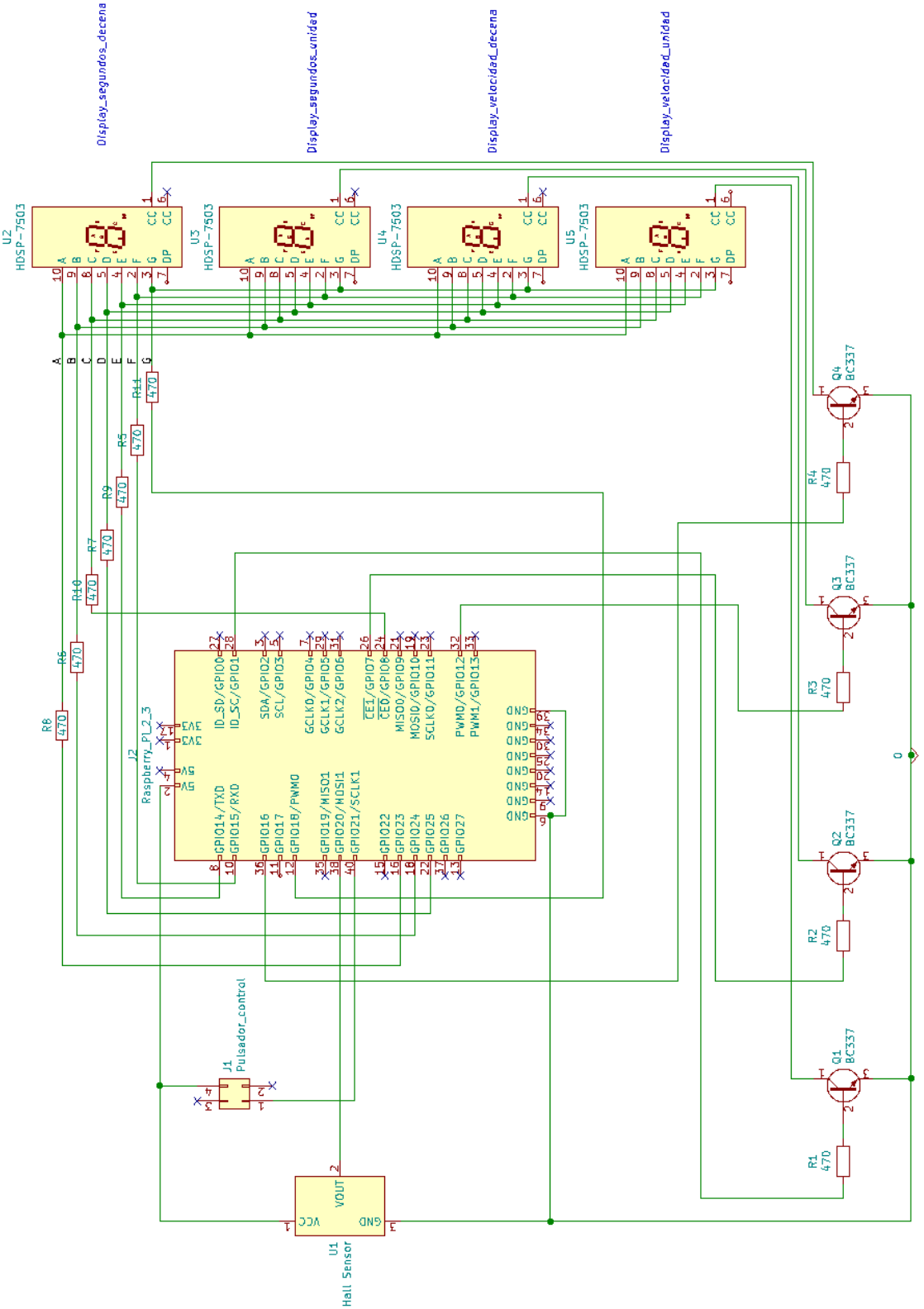
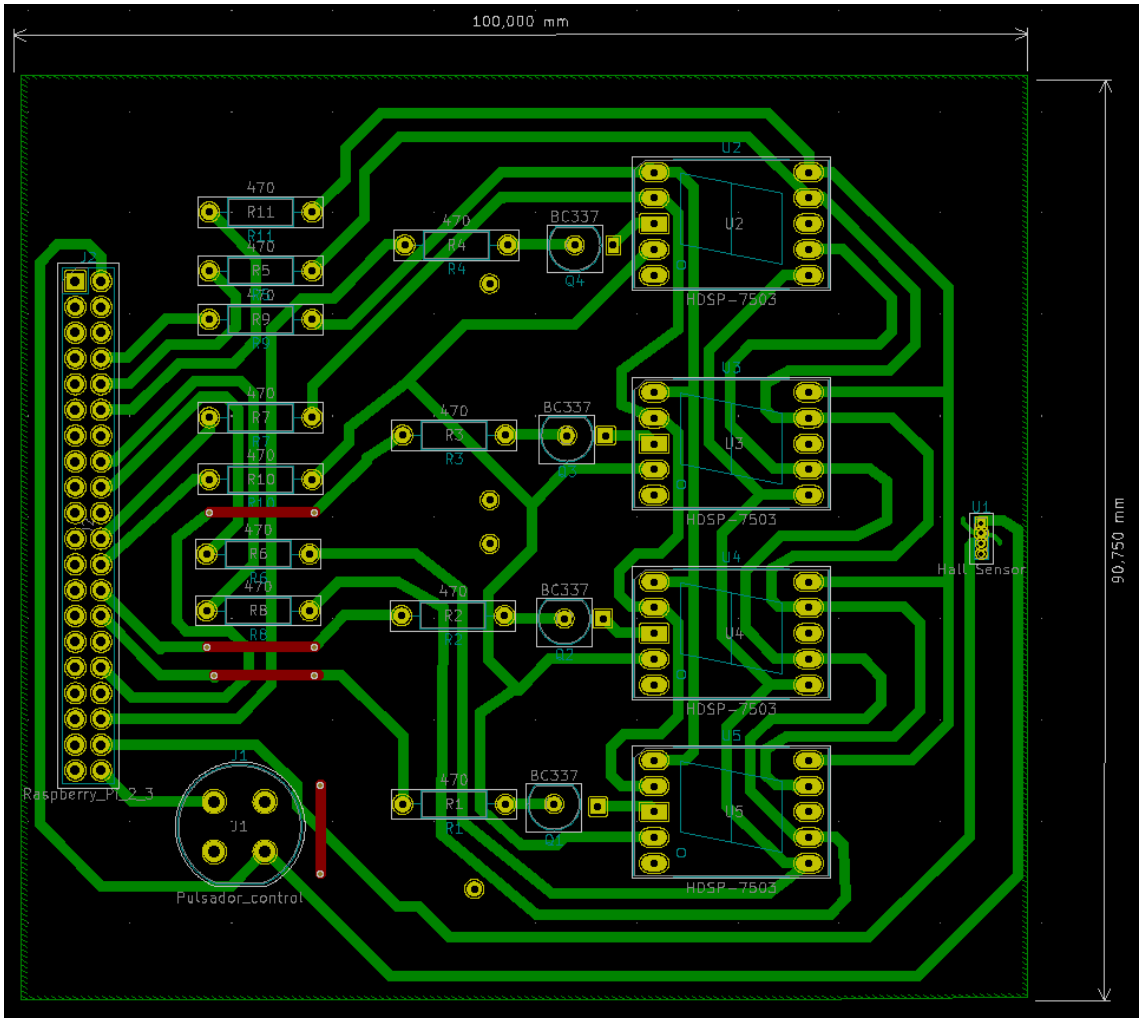


Figura 4.5 PCB del sistema en una placa de cobre



5. Pruebas

Se ejecutan las pruebas en 2 etapas. Por visualización en pantalla HDMI, y en placa.

Pantalla HDMI: Se inicia la programación del bloque de procesamiento conectando la *Raspberry Pi* a una pantalla y un teclado. Se testea posibles bugs y funcionamiento en *loop* de la programación en *Python*, **Figura 5.1**:

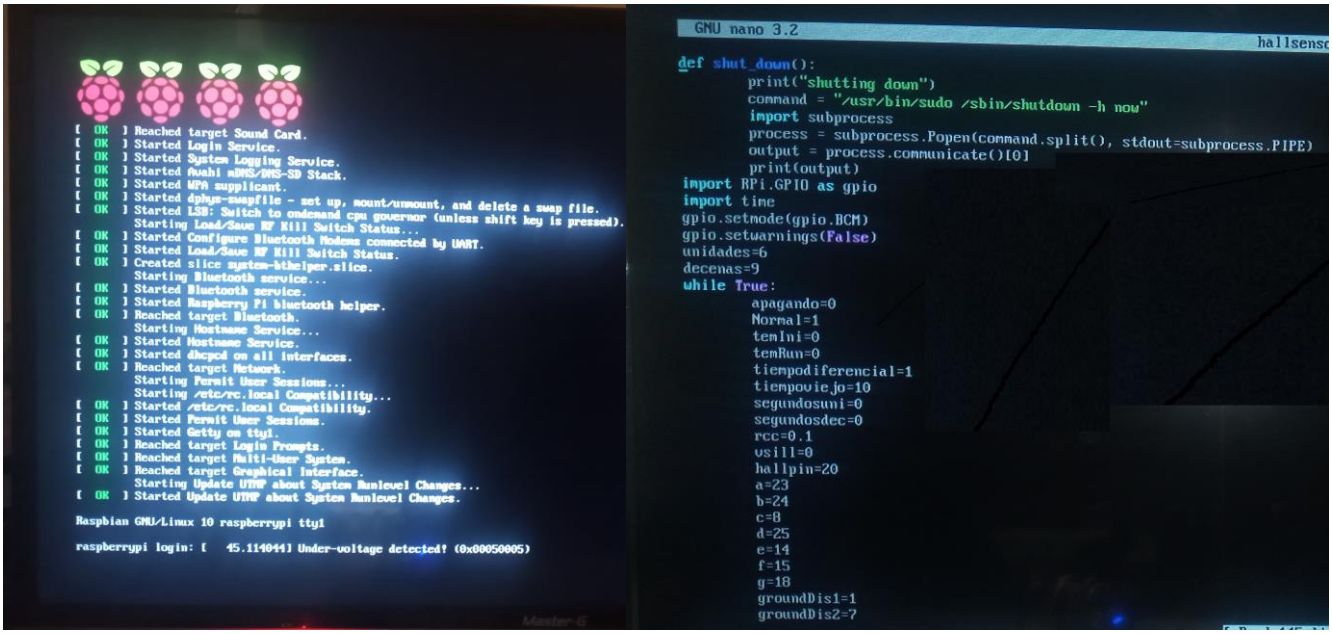


Figura 5.1 Programación de la *Raspberry Pi*

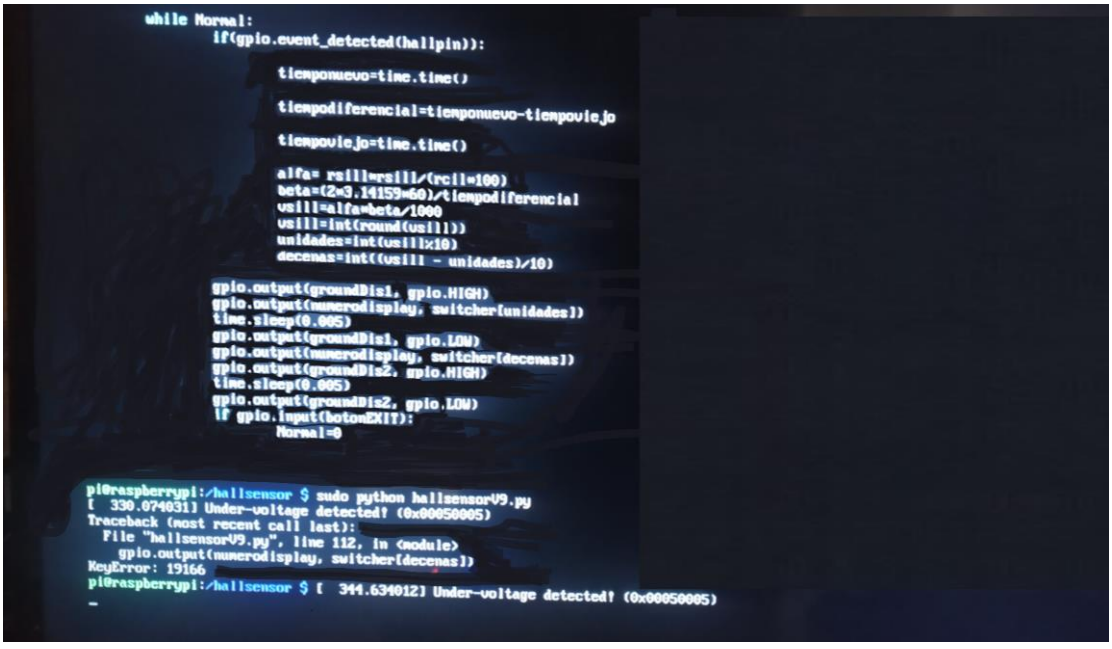


Figura 5.2 Ejemplo de bug en programación sobre la *Raspberry Pi*

La programación se realizó sobre la misma *Raspberry Pi*, para comprobar que un usuario no necesita de una computadora externa para programar el proyecto. Posible bug es presentado en la Figura 5.2.

Durante las pruebas del código aislado se producen errores de división por 0 si las ecuaciones que tengan una fracción tienen más de 6 términos en ellas, por lo que se divide en múltiples líneas de código.

Placa: Con todos los elementos del sistema funcionando como calculado, se realiza una placa para comprobar la resistencia y robustez del sistema a factores externos, Figura 5.3.

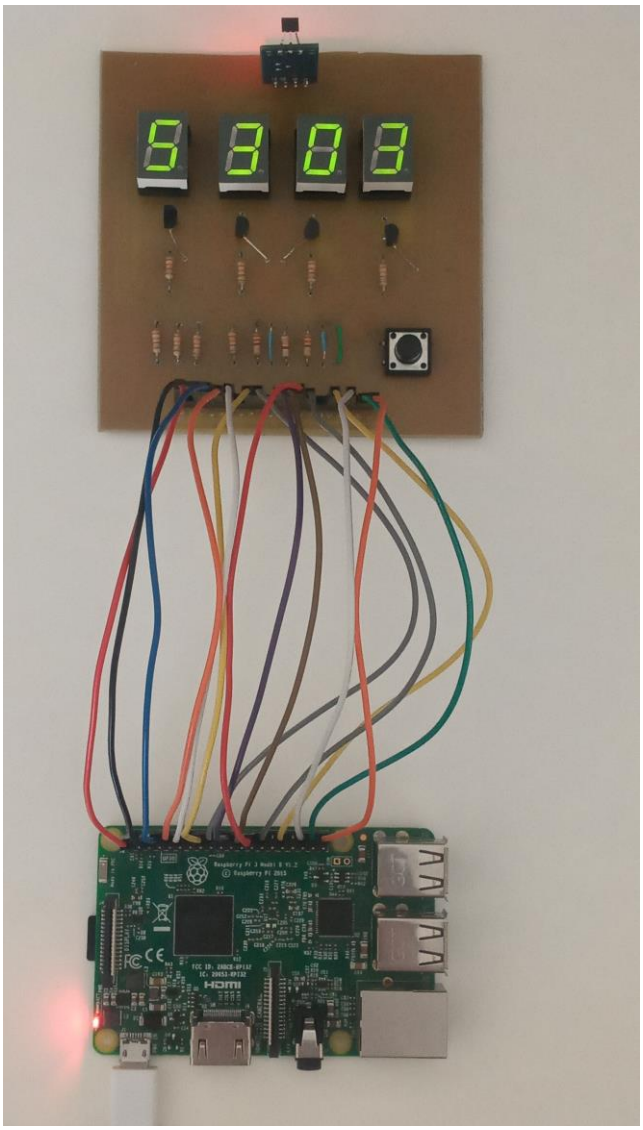


Figura 5.3 Placa de cobre visualización y control

El pulsador de control tiene una respuesta de flanco prácticamente inmediata y sin pico de tensión, Figura 5.4.

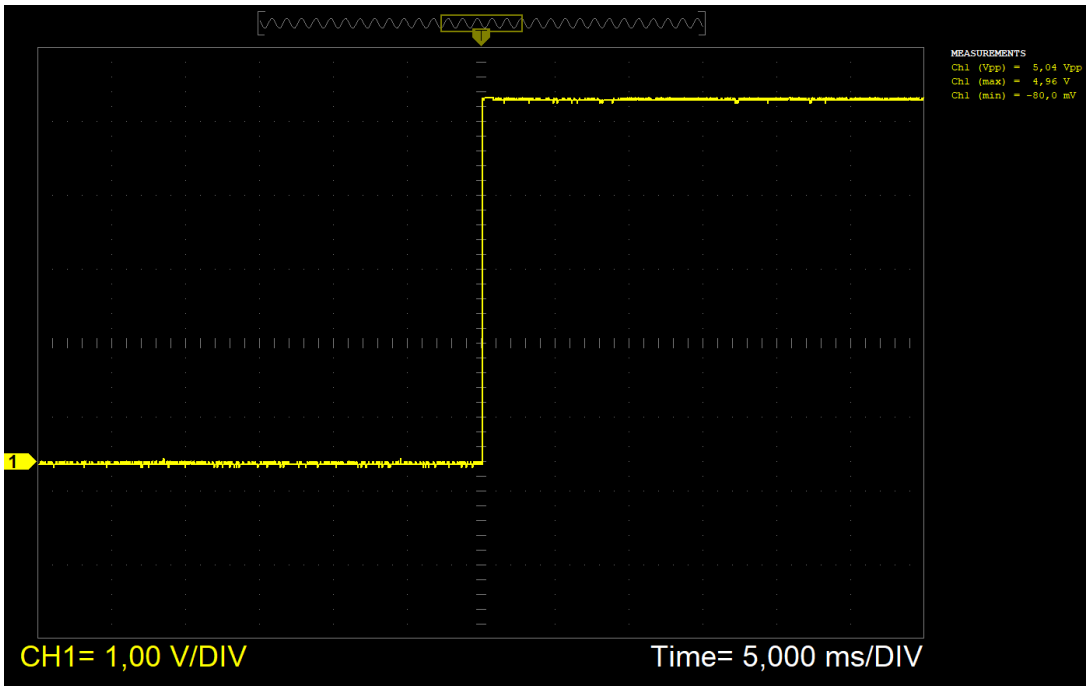


Figura 5.4 Señal de pulsador de control

En la Figura 5.5 se observa un 10% de sobre voltaje en la salida de control de los transistores de multiplexado, que dura 25ns.

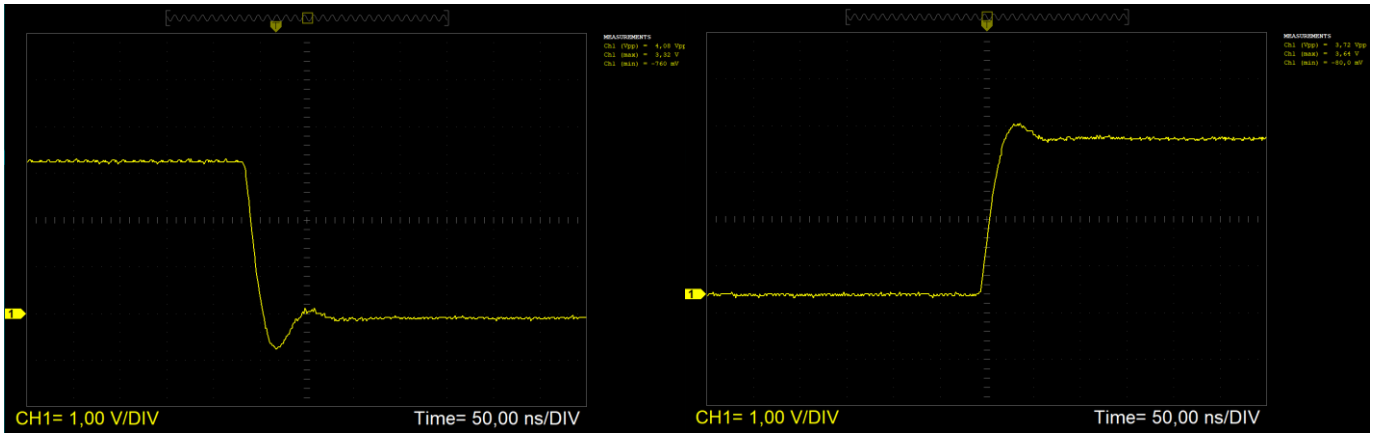


Figura 5.5 Sobre voltaje sobre los transistores de multiplexado

El voltaje de control de multiplexado en relación a los tiempos de estado apagado y prendido es visible en la Figura 5.6.

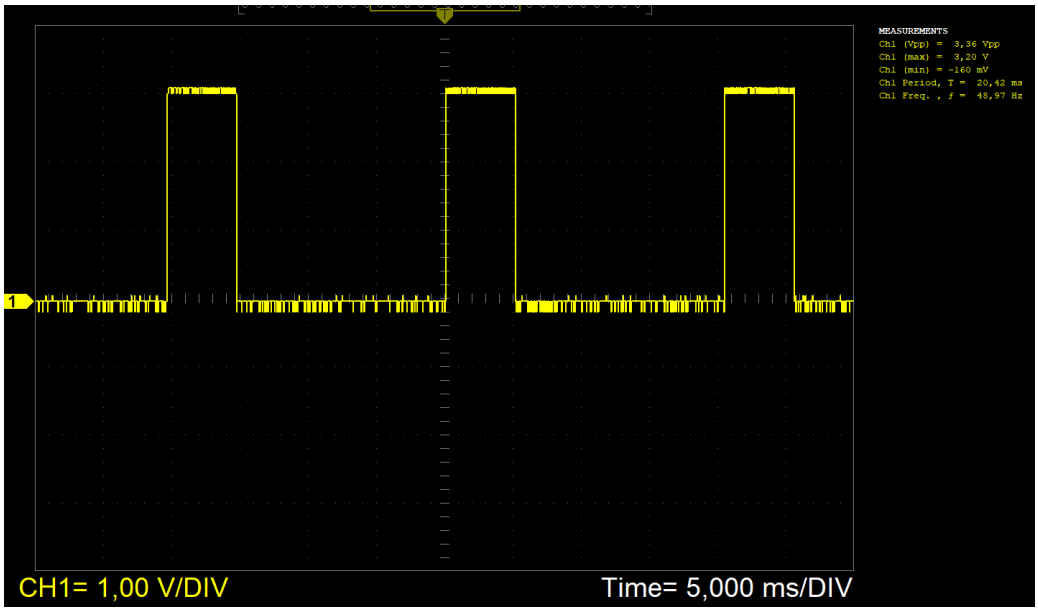


Figura 5.6 Señal de control de multiplexado

La tensión sobre dos canales de multiplexado relativos es visible en la Figura 5.7.

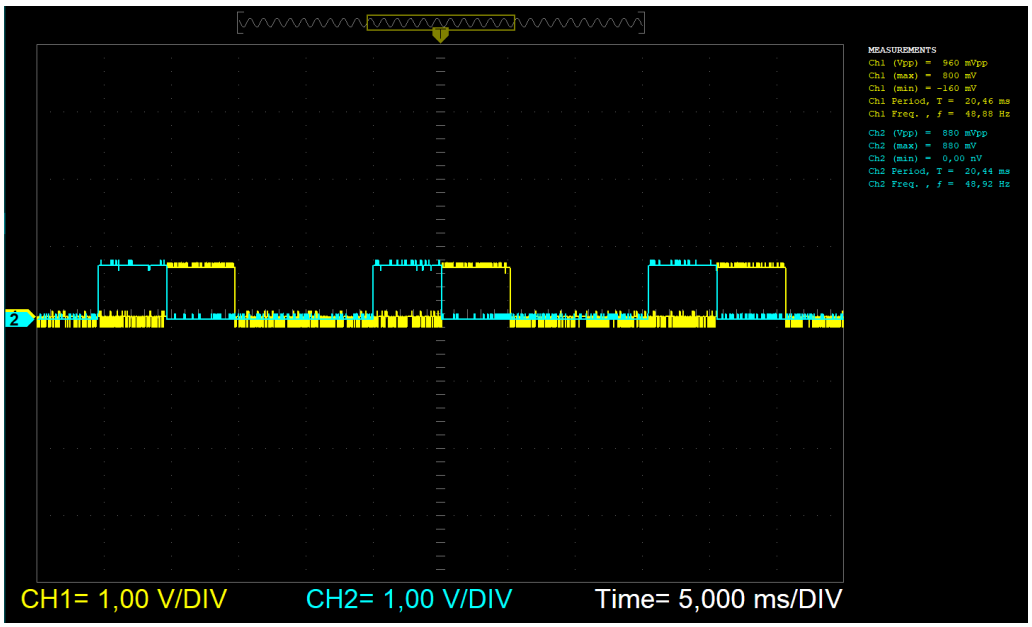


Figura 5.7 Dos señales de multiplexado superpuestas

En la Figura 5.8 se observa la curva del sensor *hall* sobre la entrada de la *Raspberry Pi*.

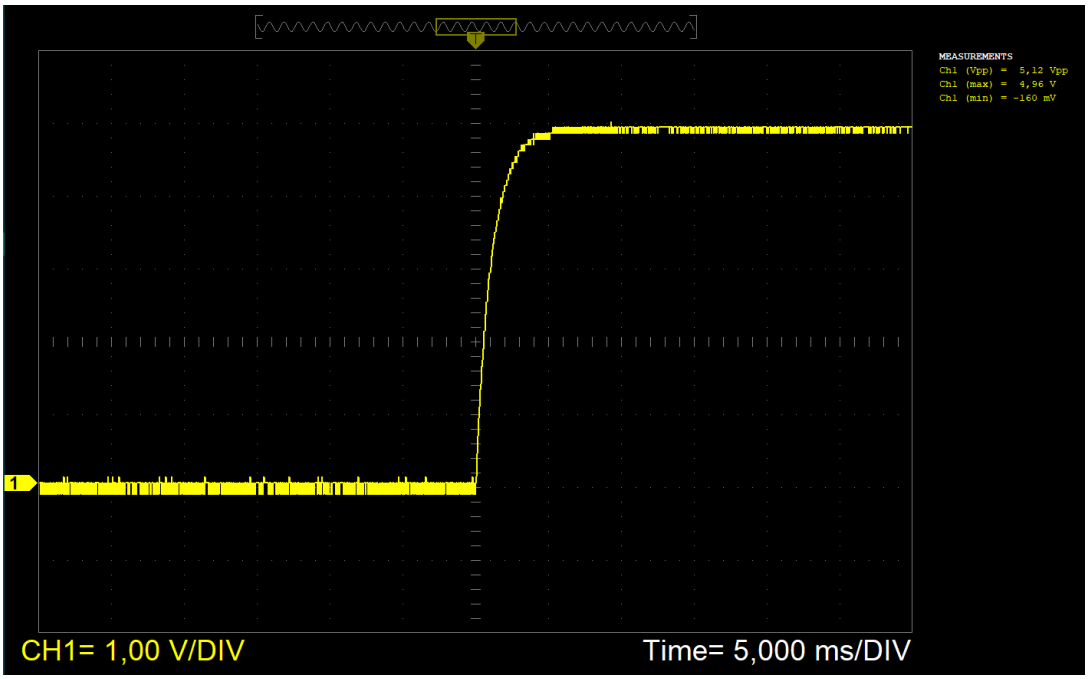


Figura 5.8 Respuesta del sensor *hall*

6. Análisis de Resultados Experimentales y Discusión

Se observa una respuesta doble en el sensor *hall* implementado ante la presencia de un campo magnético prolongado, evitable si el campo magnético circula rápido.

Para mejorar la precisión de la velocidad se puede agregar “n” imanes alrededor de los rodillos, cambiando el denominador en el cálculo de a velocidad lineal de la silla por “n”.

Aunque se observa en la Figura 5.5 ruido en los pulsos no mayor a un 10% del valor nominal, el sistema sigue respondiendo correctamente.

No se realizó pruebas del sistema en presencia de gran estática o cerca de campos electromagnéticos posibles en zonas de trabajo. Puede que la suscepción al ruido no sea alta si se conectan cables largos entre el procesamiento y los demás bloques del sistema.

En el interconexionado de la placa *Raspberry Pi* a los demás componentes del sistema se utilizan cables tipo hembra-macho Arduino, se observa que no son del todo compatibles, por lo tanto es necesario verificar cada uno de los contactos.

Conclusión

Para concluir, es verdad que se puede cambiar el bloque de procesamiento por componentes electrónicos integrados de ultra bajo costo, pero esto aumenta la cantidad de componentes significativamente. Se puede recomendar de cambiar la placa *Raspberry Pi* por una placa ESP32, ya que es posible una fácil programación en *Python*, perdiendo la posibilidad de correr un sistema operativo y tener una carpeta de *logs* para posibles errores a la hora de correr código. Al optar por otra placa de menor costo se puede perder la escalabilidad del sistema, tanto como un fácil mantenimiento de personal no especializado.

Este tipo de sistema de bloques es fácil de implementar con distintos elementos, se puede remplazar el hardware de cada bloque a medida que se cambien los requerimientos. Conservando el requerimiento de utilizar *software* y hardware libre, para seguir progresando, mejorando o cambiando este proyecto para adaptarse a otros usuarios que tengan necesidades particulares y/o diferentes.

Bibliografía

1. *Datasheet* de sensor de efecto *Hall* del módulo Arduino: https://www.datasheetarchive.com/pdf/download.php?id=20f0190f65252a349493831a8b2e5f1ea9aa74&type=P&term=3141%2520hall%2520sensor
2. *Datasheet* de *display LED* de 7 segmentos: https://www.kingbrightusa.com/images/catalog/SPEC/SC56-11SRWA.pdf
3. https://www.onsemi.com/pdf/datasheet/bc546-d.pdf
4. https://www.jameco.com/Jameco/workshop/TechTip/working-with-seven-segment-displays.html
5. https://sensing.honeywell.com/hallbook.pdf
6. https://www.mpja.com/download/a3144eul.pdf
7. https://pinout.xyz/
8. https://projects-raspberry.com/raspberry-pi-launch-python-script-on-startup-2/
9. https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/
10. https://docs.python.org/3/library/time.html
11. https://www.geeksforgeeks.org/
12. https://chmodcommand.com/chmod-755/
13. https://www.kicad.org/
14. https://www.raspberrypi.com/
15. https://lucid.app/
16. https://electronica.mercadolibre.com.ar/componentes-electronicos-raspberry-pi/
17. https://www.samsung.com/latin/mobile-accessories/power/