

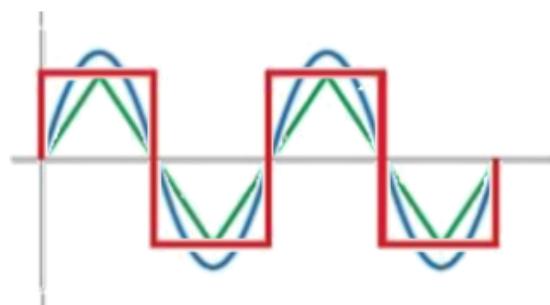
Universidad Nacional Autónoma de México  
Facultad de Ingeniería  
Microcomputadoras  
M.I. José Antonio de Jesus Arredondo Garza



# Proyecto 3. Lectura y Procesamiento de Señales

## Sistema de Seguridad

Álvarez Rodea, Carolina  
Mendoza de la Vega, Dulce Elizabeth  
Nava Rosales, Juan Manuel  
Ordiales Caballero, Iñaky



Grupo 3  
Semestre 2023-1  
18 de enero de 2023

## Objetivos

1. Desarrollar un código capaz de simular funciones de transferencia en tiempo real, haciendo uso del microprocesador Raspberry Pi Pico y de Matlab.
2. Desarrollar un proyecto libre utilizando la Raspberry Pi Pico, que demuestre los conocimientos obtenidos durante el curso a través de una aplicación práctica.

## Introducción

La *función de transferencia* es una expresión matemática que caracteriza las relaciones de “Entrada – Salida” de sistemas lineales invariantes en el tiempo. Se define como la relación relación de la transformada de Laplace de la salida (función respuesta), a la transformada de Laplace de la entrada (función excitadora), bajo condiciones iniciales en cero.

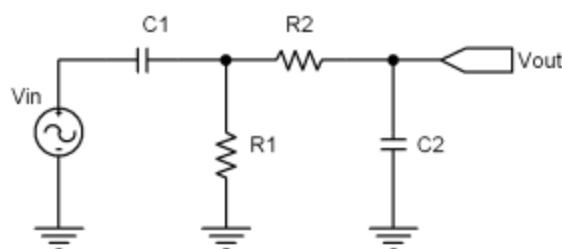
La función de transferencia se define como la siguiente relación:

$$G(s) = \frac{Y(s)}{X(s)}$$

Dicha expresión relaciona la salida y la entrada de un sistema lineal invariante en el tiempo, en términos de los parámetros del sistema.  $Y(s)$  y  $X(s)$  son denominados polinomios del numerador y del denominador respectivamente.

En el proyecto el sistema de control que se trató para obtener su función de transferencia en el dominio de la frecuencia y del tiempo fue el de un filtro pasa banda RC. Este filtro se caracteriza por dejar pasar solo un rango de frecuencias delimitada por dos frecuencias denominadas frecuencia de corte inferior y frecuencia de corte superior. Si se modifican estas frecuencias de corte, se modifica el rango de frecuencias ampliendo o disminuyendo las frecuencias que pueden pasar por él.

La disposición del filtro pasa banda es el siguiente:





En este caso la primera parte del circuito con el condensador C1 y el resistor R1 forman un filtro paso alto y la segunda parte, formada por R2 y C2, forman un filtro paso bajo; en conjunto se tiene el filtro pasa banda.

La ecuación que describe su función de transferencia y sobre la cual nos basamos en el proyecto para representarla en MATLAB, es la siguiente:

$$H(s) = HR(s) \times HC(S)$$

$$HR(S) = \frac{sRC}{sRC + 1}$$

$$HC(s) = \frac{1}{sRC + 1}$$

Donde:

## Desarrollo

Se utilizó el lenguaje de programación MatLab con la finalidad de implementar la última versión de la aplicación que el profesor nos proporcionó, además esta fue explicada y analizada a lo largo del semestre, a continuación comenzaremos por describir a grandes rasgos las herramientas lógicas y físicas necesarias para el proyecto:



Como en proyectos pasados fue necesario hacer uso de nuestra tarjeta Raspberry Pi Pico, la cual fue conectada a Matlab por medio de la UART con la finalidad de poder visualizar la ejecución del código, ahora bien, hablando sobre las herramientas lógicas y no físicas, debemos mencionar que fue necesario la creación de un filtro digital de señales analogicas.

## Explicación del Código de Python controlador AD9833

En este primer archivo del proyecto del filtro digital de señales se usó una biblioteca que encontramos en GitHub para controlar a través del protocolo SPI al módulo generador de ondas AD9833. Sin embargo, como el módulo fue desarrollado especialmente para la PyBoard, se le tuvo que hacer muy ligeras modificaciones para que funcionara con RP Pico. Además se le agregó una función a la clase AD9833, que imprimiera en pantalla la frecuencia y forma actual de la onda generada.

El código define una clase que cuenta con cinco atributos: la frecuencia de reloj, la frecuencia de onda, la forma de onda, el pin de transferencia SPI y el pin de selección SS. Además tiene 8 métodos para realizar su función. De forma resumida lo que hace es que le puedes modificar la frecuencia y forma deseada y al llamar al método send() a través de SPI se comunica con el AD9833 y cambia la configuración de sus registros para generar la onda deseada. En el código principal del programa se crea un objeto de la clase AD9833 y a través de él se accede a los métodos para controlar el generador de ondas.

## Código de Python controlador AD9833

```
#####
###*****#
####
####      Programa para controlar por SPI      ###
####          el módulo AD9833 con ayuda          ###
####          de la Raspberry Pi Pico             ###
####                                              #####
###*****#
#####

# By Kipling Crossing
# Link: https://github.com/KipCrossing/Micropython-AD9833
# Modified by Inaky Ordiales

class AD9833(object):

    # Clock Frequency
    ClockFreq = 15000000
    freq = 10000
    shape_word = 0x2000

    def __init__(self, spi, ss):
        self.spi = spi
        self.ss = ss

    # function for splitting hex into high and low bits
    def _bytes(self, integer):
        return divmod(integer, 0x100)
```



```

def _send(self, data):
    high, low = self._bytes(data)
    msg = bytearray()
    msg.append(high)
    msg.append(low)

    self.ss.value(0)
    self.spi.write(msg)
    self.ss.value(1)

def set_freq(self, freq):
    self.freq = freq

def set_type(self, inter):
    if inter == 1:
        self.shape_word = 0x2020
    elif inter == 2:
        self.shape_word = 0x2002
    else:
        self.shape_word = 0x2000

@property
def shape_type(self):
    if self.shape_word == 0x2020:
        return "Cuadrada"
    elif self.shape_word == 0x2002:
        return "Triangular"
    else:
        return "Senoidal"

def send(self):
    # Calculate frequency word to send
    word = hex(int(round((self.freq*2**28)/self.ClockFreq)))

    # Split frequency word onto its seperate bytes
    MSB = (int(word, 16) & 0xFFFFC000) >> 14
    LSB = int(word, 16) & 0xFFFF

    # Set control bits DB15 = 0 and DB14 = 1; for frequency register 0
    MSB |= 0x4000
    LSB |= 0x4000

    self._send(0x2100)
    # Set the frequency
    self._send(LSB) # lower 14 bits
    self._send(MSB) # Upper 14 bits
    # Set the shape
    # square: 0x2020, sin: 0x2000, triangle: 0x2002
    self._send(self.shape_word)

def info(self):
    # Imprime información de la onda actual
    print("\nForma: "+str(self.shape_type))
    print("Frecuencia = "+str(self.freq))

```

```
#####
###          FIN DEL CÓDIGO          ###
#####
###
```

## Explicación del Código de Python procesamiento de señales

El código principal del programa es este. Aquí se controla tanto la generación de onda, como la lectura, filtrado y envío de señales. El código está comentado de tal manera que se pueda entender su funcionamiento y en dado caso modificarlo después de revisarlo. Al principio se explica la obtención de los parámetros de filtrado conforme al filtro paso bandas que se utilizó. Primero un paso alto y después un paso bajos. Al final quedó una función de transferencia de segundo orden lo que resultó en una ecuación en diferencias de 5 constantes y variables.

De forma resumida la idea del código es la siguiente: se tiene el generador de ondas que controlamos a través de dos botones conectados a la Pi Pico, uno cambia la forma y otro la frecuencia en brincos de 25 Hz, desde 25 hasta 300 Hz. Estos botones causan interrupciones del sistema cuando son activados y mediante el manejo de interrupciones se checa primero que no sea un rebote indeseado del botón (doble clic en uno solo) y se altera una variable global indicando que alguno se presionó. En el ciclo principal se checa constantemente el estado de esta variable y en caso de cambiar su valor se manda el cambio de forma o frecuencia a través del objeto AD9833 que creamos, utilizando el protocolo de comunicación SPI.

En el ciclo principal del programa después de checar el estado de los botones, checa si se recibió algún carácter por el objeto de comunicación UART. En caso afirmativo checa si es la llave deseada ('U') o si es cualquier otro carácter. Sólo si es la llave correspondiente empieza con la toma de muestras usando el objeto del convertidor analógico digital ADC que declaramos en un inicio. Toma 1000 muestras a una frecuencia de muestreo de 8 kHz y cada vez que toma una muestra usa la ecuación de diferencias para aplicar el filtro digital y general la señal de salida que se almacena. Actualiza los valores de la ecuación y vuelve a tomar otra muestra hasta llegar a 1000. Ya que se tienen todas las señales de entrada y de salida, se empaquetan en un byte-array donde cada muestra ocupa 4 bytes y se envían a través del objeto de comunicación UART. Las muestras de entrada y salida se intercalan para evitar un retraso indeseado en las señales.

# Código de Python procesamiento de señales

```
#####
##### sys = tf ( [0.0016 0], [0.00000128 0.0024 1] )
#####
#####          0.0016 s
#####
##### sys = -----
#####      1.28e-06 s^2 + 0.0024 s + 1
#####
#####
##### sysd = c2d (sys, 0.000125, 'tustin')
#####
#####          0.06974 z^2 - 0.06974
#####
##### sysd = -----
#####      z^2 - 1.78 z + 0.7908
#####
#####
#####
##### ECUACION EN DIFERENCIAS:
#####
#####
##### Vo(k) =  0.06974Vi(k) + 0Vi(k-1) - 0.06974Vi(k-2) +
#####
#####           + 1.78Vo(k-1) - 0.7908Vo(k-2)
#####
#####
#####
##### UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
#####
##### FACULTAD DE INGENIERÍA
#####
##### MICROCOMPUTADORAS
#####
#####
#####
#####
```

```
# Importamos las bibliotecas necesarias
from machine import Pin      # Usar pines Pi Pico
from machine import UART     # Protocolo UART para MatLab
from machine import SPI       # Protocolo SPI para ad9833
from machine import ADC       # Convertidor analógico-digital
from ad9833 import AD9833   # Estructura para controlar el AD9833
from time import sleep       # retardo en s
from time import sleep_us    # retardo en us
from time import ticks_ms    # herramienta de tiempo
import struct                # convertir de float a hexadecimal
```

```
#####
### #####
### CONFIGURACIONES INICIALES #####
### #####
#####
```

```
# Hacer que el CPU trabaje a 24 MHz
machine.freq(240000000)
```

```
# Declarar convertidor ADC por canal 0
adc0 = ADC(0)

# Declara UART para comunicarse con MATLAB
uart0 = UART(0, baudrate=460800, tx=Pin(0), rx=Pin(1))

# Usamos led de la tarjeta como registro
P25 = Pin(25, Pin.OUT)

#####
### CONFIGURACIONES AD9833 ###
#####
###

# Declarar SPI para controlar el ad9833
cs = Pin(17, Pin.OUT) # activación esclavo spi
spi0 = SPI( 0,
            baudrate=9600,
            polarity=1,
            phase=1,
            bits=8,
            firstbit=machine.SPI.MSB,
            sck=machine.Pin(18),
            mosi=machine.Pin(19),
            miso=machine.Pin(16))

# Variables para el control de la onda (AD9833)
frecuencia = 0
forma = 0
debounce_time = 0
botonpres = 0

# Botones para el control de la señal (AD9833)
botonForma = Pin(14, Pin.IN, Pin.PULL_UP)
botonFrec = Pin(15, Pin.IN, Pin.PULL_UP)

# Inicialización de la señal del AD9833
wave = AD9833(spi0, cs)
wave.set_freq(frecuencia+25)
wave.set_type(forma+1)
wave.send()
sleep(0.5)
wave.info()
sleep(0.5)

#####

### CONFIGURACIONES INTERRUPCIONES ###
#####
### al apretar un botón se lleva una #####
### interrupción para modificar la #####
#####
```



```

### onda, cuidando que no haya un      ###
### rebote al apretarlo.            ###
###                                     ###
###                                     ###
##########
# Definición interrupciones onda
def callback(botonForma):
    global botonpres, debounce_time
    if (ticks_ms() - debounce_time) > 250:
        debounce_time = ticks_ms()
        botonpres = 1

botonForma.irq(trigger=Pin.IRQ_FALLING, handler=callback)

def callback(botonFrec):
    global botonpres, debounce_time
    if (ticks_ms() - debounce_time) > 250:
        debounce_time = ticks_ms()
        botonpres = 2

botonFrec.irq(trigger=Pin.IRQ_FALLING, handler=callback)

#####
###                                     ###
### Variables para lectura de señales ###
###                                     ###
#####

# Declaración de variables y constantes usadas
muestras = 1000
entradas = [0] * muestras
salidas = [0] * muestras
conv_bytes = bytearray(4*2*muestras)
factor = 3.3/(65535) # Factor de conversión de digital a voltaje

# Coeficientes de la Función de Transferencia
CTE_A0 = 0.06974
CTE_A1 = 0.0
CTE_A2 = -0.06974
CTE_B1 = 1.78
CTE_B2 = -0.7908

# Condiciones iniciales de las entradas y salidas
UK    = 0.0 # u(k)
UK1   = 0.0 # u(k-1)
UK2   = 0.0 # u(k-2)
YK    = 0.0 # y(k)
YK1   = 0.0 # y(k-1)
YK2   = 0.0 # y(k-2)
Leer  = 0.0

```

```
#####
### Código principal del programa #####
### LECTURA Y PROCESAMIENTO DE LAS SEÑALES #####
#####

while True:

    # Checa si se presionó para cambiar la forma
    if botonpres == 1:
        forma = (forma + 1) % 3
        wave.set_type(forma + 1)
        wave.send()
        wave.info()
        botonpres = 0

    # Checa si se presionó para cambiar la frecuencia
    if botonpres == 2:
        frecuencia = (frecuencia + 25) % 300
        wave.set_freq(frecuencia + 25)
        wave.send()
        wave.info()
        botonpres = 0

    if uart0.any() > 0:

        LLAVE = uart0.read(1)      # Lee 1 byte recibido

        # Se espera a que se ingrese la llave correspondiente
        if "U" in LLAVE:
            P25.on()

        for i in range(0,muestras):

            # Armado de la ecuación en diferencias
            YK = CTE_A0*UK + CTE_A1*UK1 + CTE_A2*UK2 + CTE_B1*YK1 + CTE_B2*YK2

            # Actualiza I/O para la próxima iteración
            UK2 = UK1  # u(k-1) --> u(k-2)
            UK1 = UK   # u(k)   --> u(k-1)
            YK2 = YK1  # y(k-1) --> y(k-2)
            YK1 = YK   # y(k)   --> y(k-1)

            # Lectura de la señal digital y conversión a valor analógico
            Leer = adc0.read_u16() * factor

            UK = Leer
            entradas[i] = UK
            salidas[i] = YK+1.65 # se compensa el offset de la salida
```

```
sleep_us(80)

P25.off()

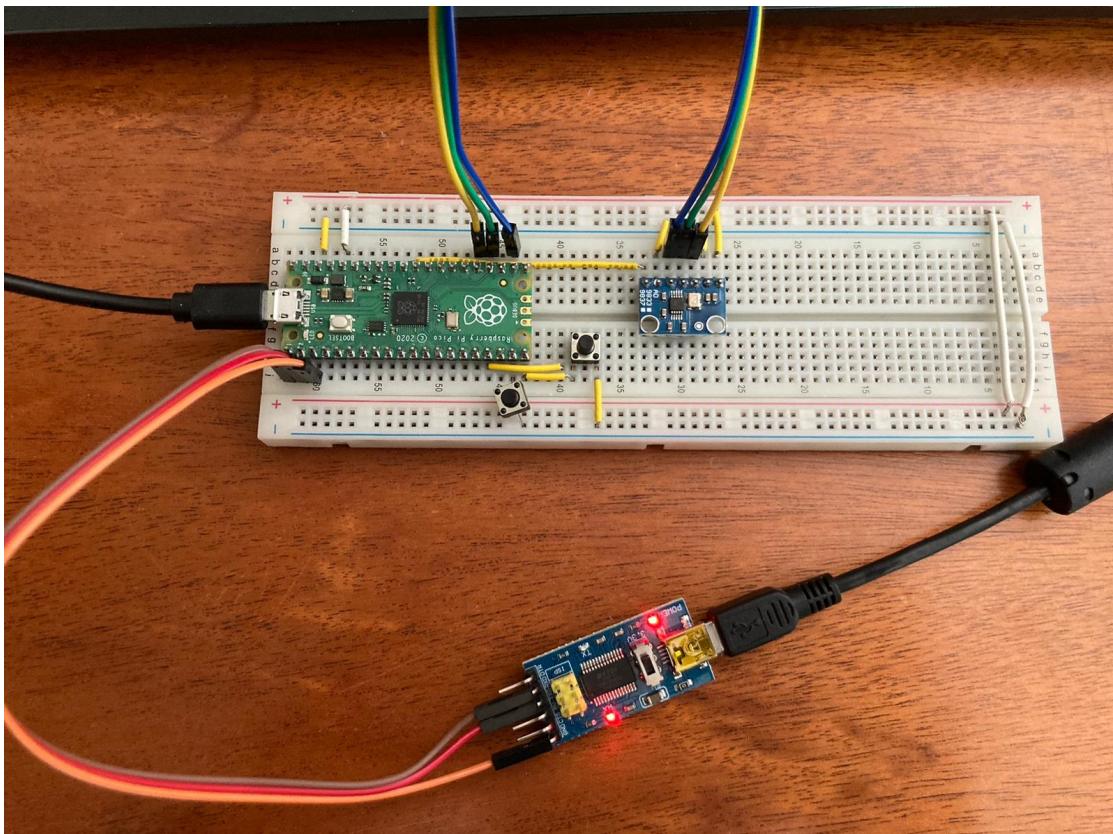
# Empaqueamiento de los datos en byte-arrays
conv_bytes = bytearray(struct.pack("f", entradas[0]))
conv_bytes = conv_bytes + bytearray(struct.pack("f", salidas[0]))

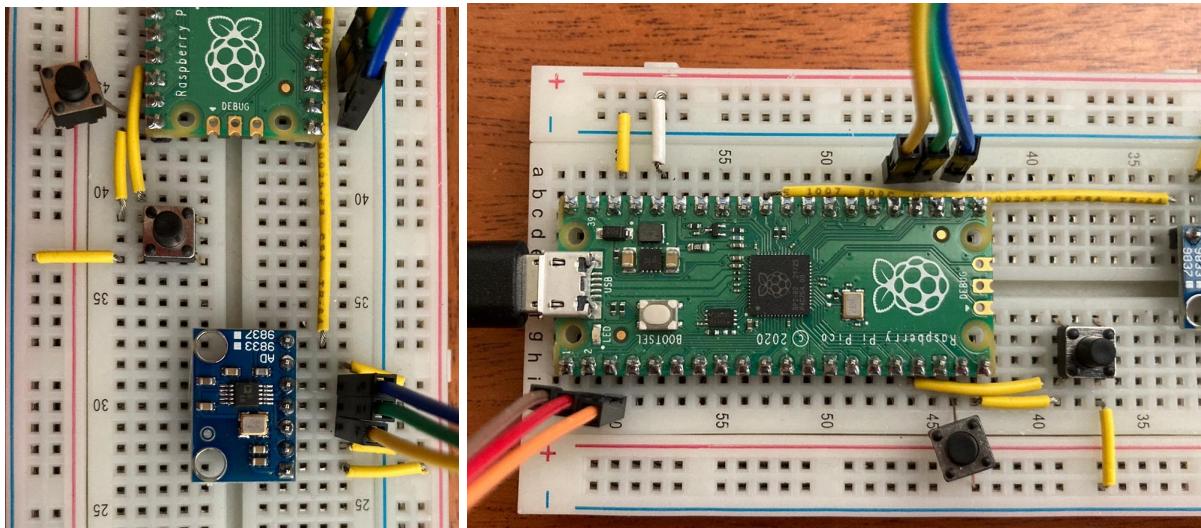
for i in range(1,muestras):
    conv_bytes = conv_bytes + bytearray(struct.pack("f", entradas[i]))
    conv_bytes = conv_bytes + bytearray(struct.pack("f", salidas[i]))

# Envío de los datos
uart0.write(conv_bytes)

#####
### FIN DEL PROGRAMA ###
#####
```

## Ensamblado del circuito





## Evidencias de la ejecución

### Inicio programa:

```

[ main.py ] x
 9 #####
10 #####
11 #####
12 ##### Simulador de un circuito paso banda en la RASPBERRY PI PICO #####
13 #####
14 #####
15 #####
16 #####          C1           R2           #####
17 #####          -----|-----/\/\/\-----#####
18 #####          +           |           |       +#####
19 #####          >           |           |       #####
20 #####          Vi(t)      R1 <       C2 =   Vo(t) #####
21 #####          >           |           |       #####
22 #####          -           |           |       -#####
23 #####          -----|-----#####
24 #####
25 #####
26 #####
27 #####
Shell x

MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

Forma: Cuadrada
Frecuencia = 25

```

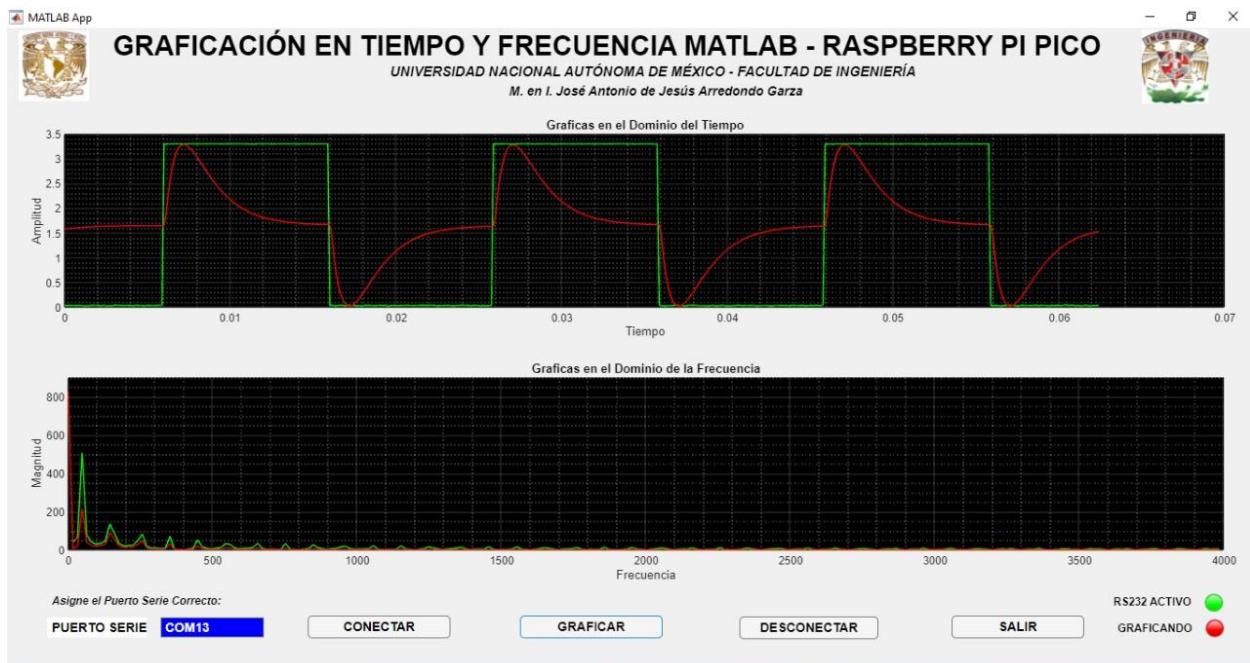
## Cambio forma y frecuencia:

```
[ main.py ] <
9 #####
10 #####
11 #####
12 ##### Simulador de un circuito paso banda en la RASPBERRY PI PICO #####
13 #####
14 #####
15 #####
16 #####          C1          R2          #####
17 #####          -----|-----/\/\/\----- #####
18 #####          +           |           +   #####
19 #####          >           |           >   #####
20 #####      Vi(t)     R1 <     C2 = Vo(t) #####
21 #####          >           |           -   #####
22 #####          -           |           -   #####
23 #####
24 #####
25 #####
26 #####
27 #####
Shell <
Peticionaria ->
Forma: Triangular
Frecuencia = 75

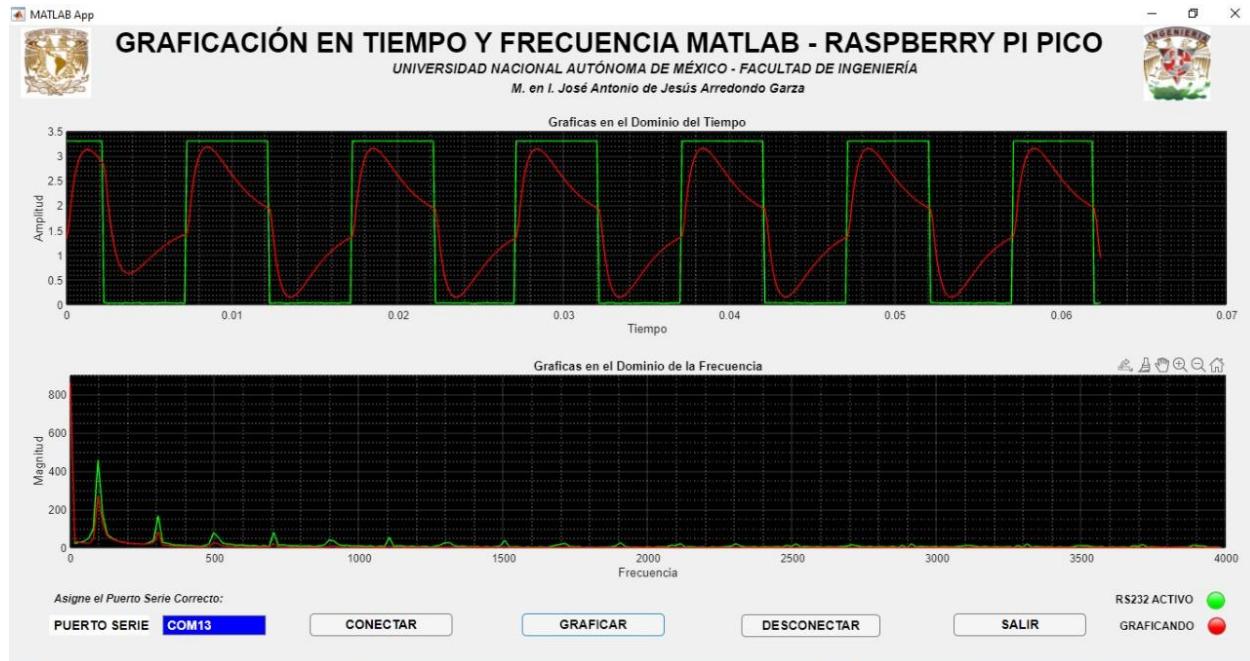
Forma: Senoidal
Frecuencia = 75

Forma: Cuadrada
Frecuencia = 75
```

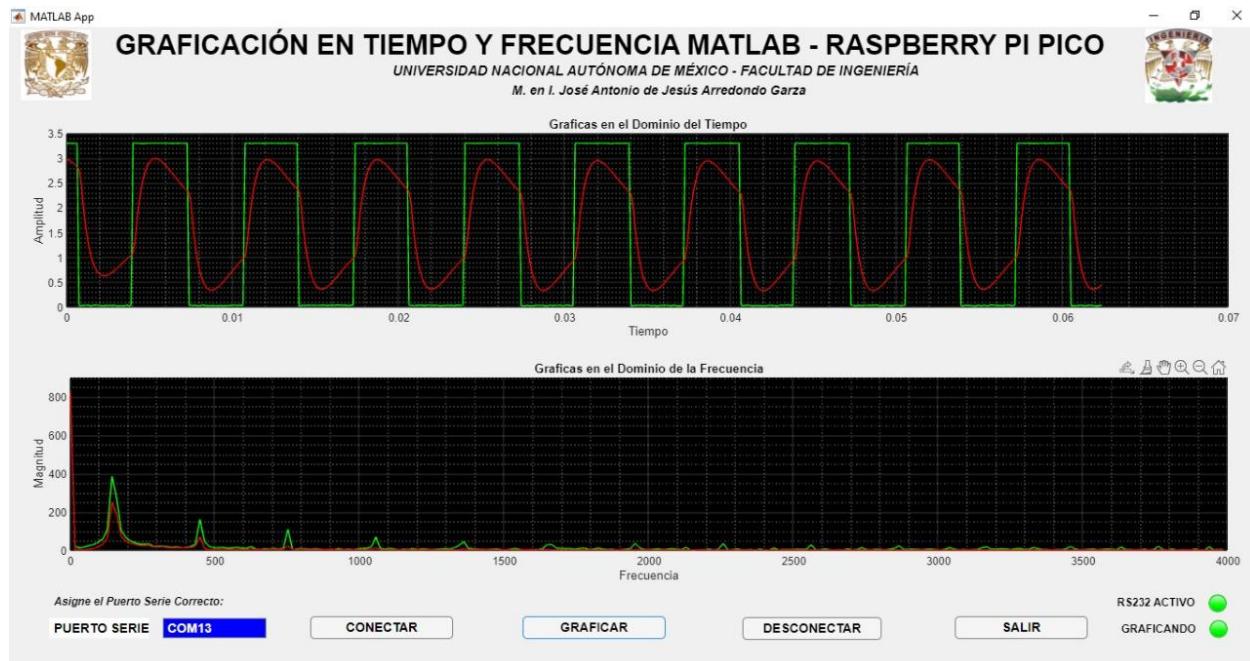
Con señal de entrada cuadrada de 50 Hz.



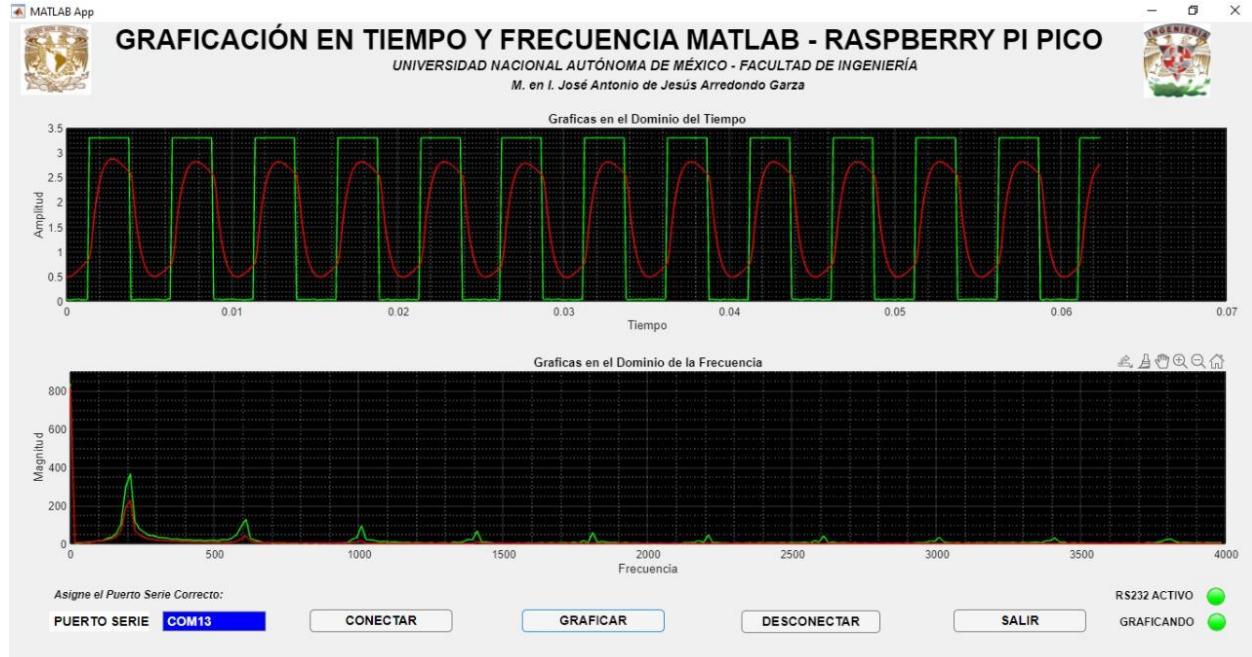
Con señal de entrada cuadrada de 100 Hz.



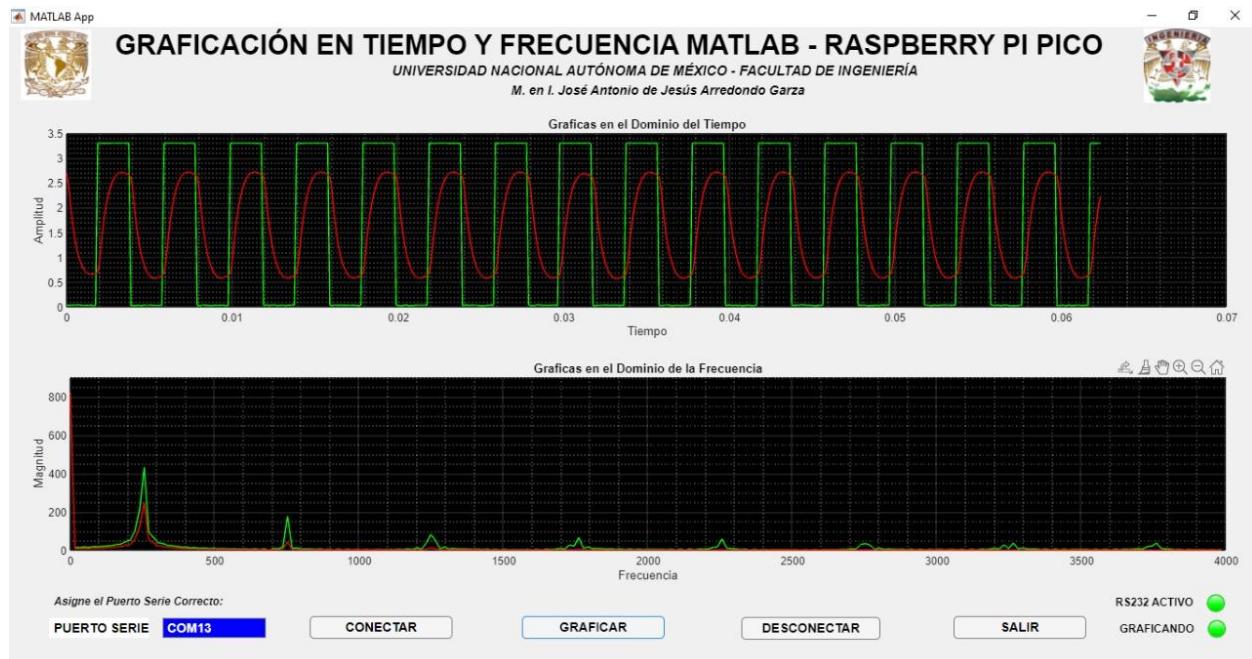
Con señal de entrada cuadrada de 150 Hz.



Con señal de entrada cuadrada de 200 Hz.

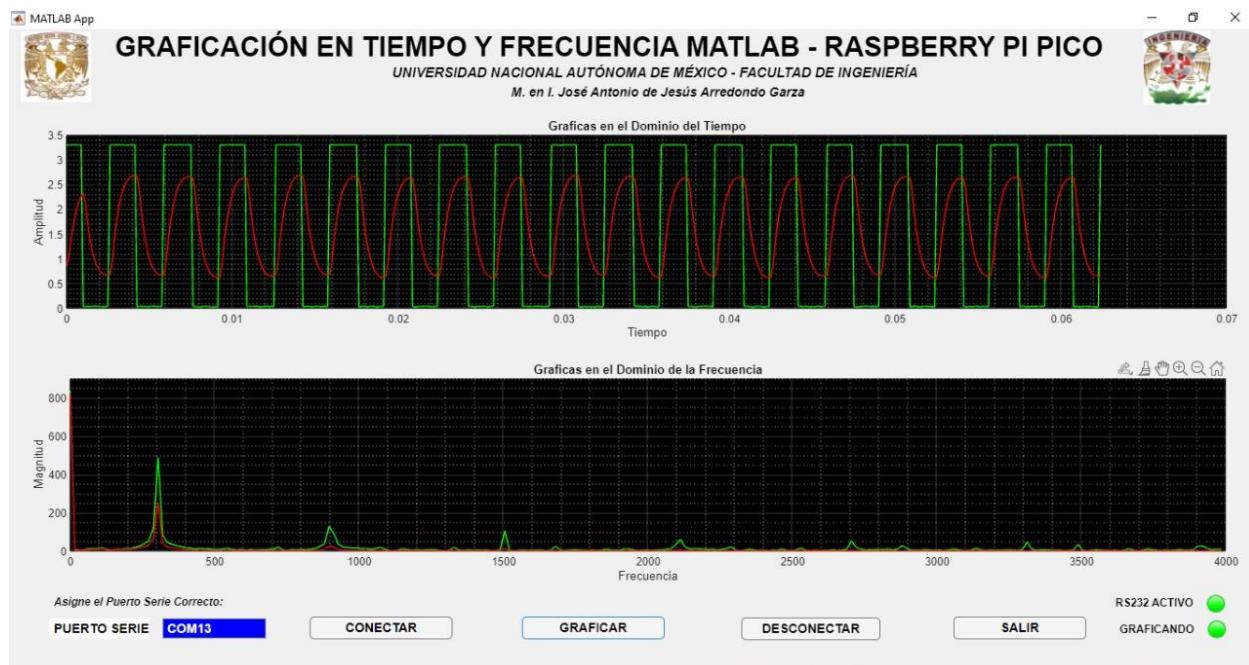


Con señal de entrada cuadrada de 250 Hz.





Con señal de entrada cuadrada de 300 Hz.



## PROYECTO ADICIONAL - Simulación de un sistema de seguridad



### Descripción

El proyecto consiste en simular un sistema de seguridad, dicho sistema se encarga de validar si un objeto está lo suficientemente cerca del sistema y entonces bloquearse. Para la implementación se usó un servomotor para controlar las pinzas que abren y cierran según la condición de distancia y un sensor ultrasónico para verificar la distancia de los objetos que se acercan al sistema.

En nuestra implementación si algo se acerca al sistema a menos de 10 centímetros, el sistema se bloquea, esto es, se cierran las pinzas y brilla el led rojo como alarma. El sistema solo se desbloquea hasta que se presione el botón.

### Desarrollo

#### Explicación del Código de Python sistemaSeguridad

En este programa opcional se utilizó PWM (Pulse Width Modulation) para controlar la intensidad de Leds y la posición del Servomotor que simulaba el sistema de seguridad que se abre y se cierra. También se usaron las interrupciones para que en cualquier momento en el que se apriete el botón de liberación, se maneje su lógica correspondiente. Además, se usó un sensor ultrasónico para determinar la distancia y bloquear todo cuando algo se acercara demasiado. Y muy importante de mencionar, es que se usaron dos hilos de ejecución para aprovechar el paralelismo que nos ofrecen los dos núcleos de la Raspberry Pi Pico.

Es relevante el mencionar que como se está trabajando en multinúcleo de forma paralela, se deben cuidar las condiciones de carrera para el acceso y modificación de variables globales y compartidas. Para esto se pueden utilizar los famosos candados o semáforos, pero se nos hizo más sencilla la solución de controlar la condición de carrera a través del flujo lógico de nuestro programa. Entonces se dividieron las tareas que el sistema lleva a cabo en dos partes:

- En el núcleo principal, primer núcleo, primer hilo de ejecución: se realiza la lectura y procesamiento de distancia con el sensor ultrasónico. Cuando la condición de cercanía se cumple se cierra la pinza (activa alarma) y deja de calcular la distancia. Hasta el momento en que se desactive la alarma manualmente vuelve a abrir la pinza y realiza las lecturas de distancia. Cada vez que mide una distancia envía un mensaje a la computadora al respecto a través del cable USB que lo conecta a la computadora.
- En el segundo núcleo o segundo hilo de ejecución: se realiza una rutina con leds constantemente, a un ritmo independiente del primer núcleo y en cuanto se activa la alarma prende el led rojo y se pone en espera de la variable de que se presionó la liberación del sistema (apagar la alarma). Mientras no se apriete el botón, seguirá bloqueado esperando. Además éste núcleo envía al log de la computadora el mensaje de activación o desactivación, junto con el número de veces que se ha activado la alarma. La secuencia de leds es una luz fuerte que pasa de un lado a otro y se va desvaneciendo.

## Código de Python sistemaSeguridad

```
#####
###*****#
####
###    PROYECTO FINAL: SISTEMA DE SEGURIDAD    ###
####
####
###    Se simula un sistema de seguridad que se    ###
###    bloquea en cuanto detecta un intruso cerca    ###
###    del sensor. Está implementado en dos núcleos    ###
###    y se comunica vía USB para dejar un Log en la    ###
###    computadora de registro.                      ###
####
###*****#
#####
```

```
# Importamos las bibliotecas que usaremos
from machine import Pin      # Usar pines Pi Pico
from machine import PWM       # Usar Pulse Width Modulation en la Pi Pico
from time import ticks_ms    # Herramienta de tiempo para evitar rebotes
import _thread               # Hilo para permitir trabajar con dos núcleos
import utime                 # Biblioteca para retardos

#####
##### XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX #####
#####
##### Declaración de las Funciones #####
##### de los Pines. #####
#####
##### XXXXXXXXXXXXXXXXXXXXXXXXX #####
#####

# Para el sensor Ultrasónico
Trigger = Pin(26,Pin.OUT)
Echo     = Pin(27,Pin.IN, Pin.PULL_DOWN)

# Para botón desbloqueo
Boton   = Pin(15, Pin.IN, Pin.PULL_UP)

# Para el servo
pwmServo = PWM(Pin(28))
pwmServo.freq(50)

# Para los leds
LedRojo = Pin(21, Pin.OUT)
pwmLed1  = PWM(Pin(20))
pwmLed2  = PWM(Pin(19))
pwmLed3  = PWM(Pin(18))
pwmServo.freq(50)
pwmLed1.freq(1000)
pwmLed2.freq(1000)
pwmLed3.freq(1000)

# Valores PWM para el servo
abierto  = 4000 # valor para abrir el servo
cerrado  = 6350 # valor para cerrar el servo

# Variables utilizadas
alarma = 0
cuenta = 0
debounce_time = 0
botonpres = 0

# Valores iniciales
pwmServo.duty_u16(abierto)
pwmLed1.duty_u16(7000)
pwmLed1.duty_u16(7000)
```

```
pwmLed1.duty_u16(7000)
LedRojo.high()

#####
### CONFIGURACIONES INTERRUPCIONES ###
### al apretar un botón se lleva una #####
### interrupción para liberar el #####
### servo en caso de estar cerrado, #####
### cuidando que no haya un rebote #####
###

# Definición interrupción botón
def callback(Boton):
    global alarma, botonpres, debounce_time
    if (ticks_ms() - debounce_time) > 250:
        debounce_time = ticks_ms()
        if alarma == 1:
            botonpres = 1

Boton.irq(trigger=Pin.IRQ_FALLING, handler=callback)

#####
### Función para leer distancia #####
###

def read():
    Trigger.low()
    utime.sleep(0.1)

    # avienta ondas por 4us
    Trigger.high()
    utime.sleep_us(4)
    Trigger.low()

    # calcula el tiempo inicial
    while Echo.value() == 0:
        tiempoInicio = utime.ticks_us()

    # calcula el tiempo final
    while Echo.value() == 1:
        tiempoFin = utime.ticks_us()

    # hace la conversión entre tiempo y distancia según la
    # velocidad de las ondas.
    tiempo = tiempoFin - tiempoInicio
    distancia = (tiempo * 0.043) / 2
    distancia = round(distancia, 0)
```

```
# regresa como resultado la distancia en cm
return distancia

#####
### SEGUNDO NÚCLEO #####
#####

def segundoNucleo():
    global alarma, cuenta, botonpres, LedRojo, pwmLed1, pwmLed2, pwmLed3
    # Valores PWM
    prendido = 7000 # valor para prender led
    medioAlto = 3000 # valor para poner led medio alto
    medioBajo = 1000 # valor para poner led medio bajo
    apagado = 0 # valor para apagar led

    secuencia = 0 # identificador inicial secuencia leds
    ante = 0 # controlar si se acaba de cerrar

    # asignación valores iniciales
    valor1 = apagado
    valor2 = apagado
    valor3 = apagado
    LedRojo.low()

    # ciclo infinito, programa principal segundo núcleo
    while True:

        utime.sleep(0.2)

        # si hay alarma no mueve leds, checa por el botón de liberación.
        if alarma == 1:
            if ante == 0:
                LedRojo.high()
                pwmLed1.duty_u16(apagado)
                pwmLed2.duty_u16(apagado)
                pwmLed3.duty_u16(apagado)
                print("\n!!! Se ha activado la alarma !!!")
                print("Es la vez número: "+str(cuenta)+"\n")
                ante = 1
            if botonpres == 1:
                LedRojo.low()
                alarma = 0
                botonpres = 0
                print("\nSe ha desactivado la alarma.\n")
                ante = 0
            continue

        # Si no está activada la alarma hace rutina con leds
        pwmLed1.duty_u16(valor1)
```

```
pwmLed2.duty_u16(valor2)
pwmLed3.duty_u16(valor3)
secuencia += 1

if secuencia == 1:
    valor3 = valor2
    valor2 = valor1
    valor1 = prendido
elif secuencia == 2:
    valor3 = valor2
    valor2 = valor1
    valor1 = medioAlto
elif secuencia == 3:
    valor3 = valor2
    valor2 = valor1
    valor1 = medioBajo
elif secuencia == 4:
    valor3 = valor2
    valor2 = valor1
    valor1 = apagado
elif secuencia == 5:
    valor3 = valor2
    valor2 = valor1
    valor1 = apagado
elif secuencia == 6:
    valor3 = valor2
    valor2 = valor1
    valor1 = apagado
elif secuencia == 7:
    valor1 = valor2
    valor2 = valor3
    valor3 = prendido
elif secuencia == 8:
    valor1 = valor2
    valor2 = valor3
    valor3 = medioAlto
elif secuencia == 9:
    valor1 = valor2
    valor2 = valor3
    valor3 = medioBajo
elif secuencia == 10:
    valor1 = valor2
    valor2 = valor3
    valor3 = apagado
elif secuencia == 11:
    valor1 = valor2
    valor2 = valor3
    valor3 = apagado
elif secuencia == 12:
    valor1 = valor2
    valor2 = valor3
    valor3 = apagado
secuencia = 0
```

```
#####
###                                     ###
##          FIN DEL PROGRAMA DEL SEGUNDO NÚCLEO          ###
##                                     ###
#####
#####
###*****#
#####  Código principal del programa  #####
#####  #####
###*****#
#####
# Iniciamos el hilo de ejecución para el segundo núcleo
_thread.start_new_thread(segundoNucleo, ())

# Variable para saber cuando abrir
ante = 0

# Hilo principal, primer núcleo
while True:

    # si está activada la alarma no hace nada
    if alarma == 1:
        continue

    # si por primera vez se cicla tras desactivar, se abre
    if ante == 1:
        pwmServo.duty_u16(abierto)
        ante = 0

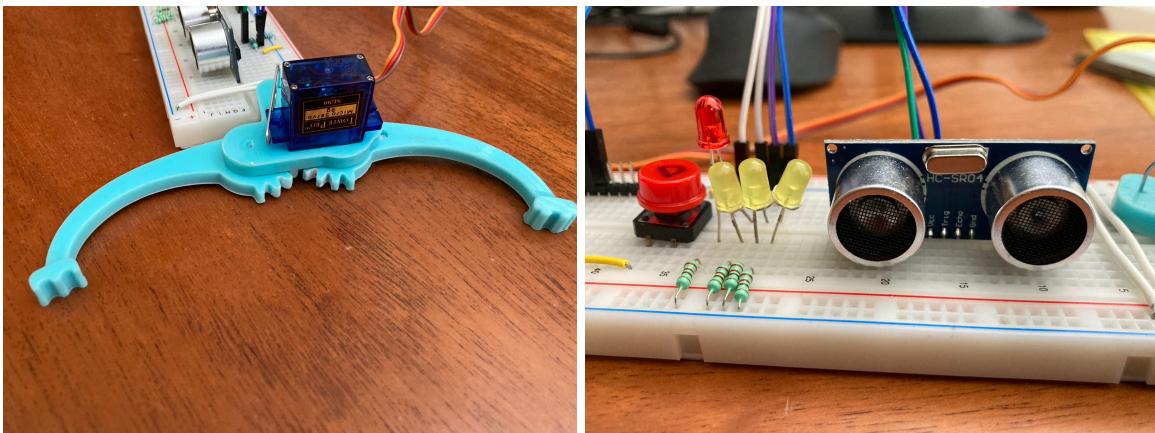
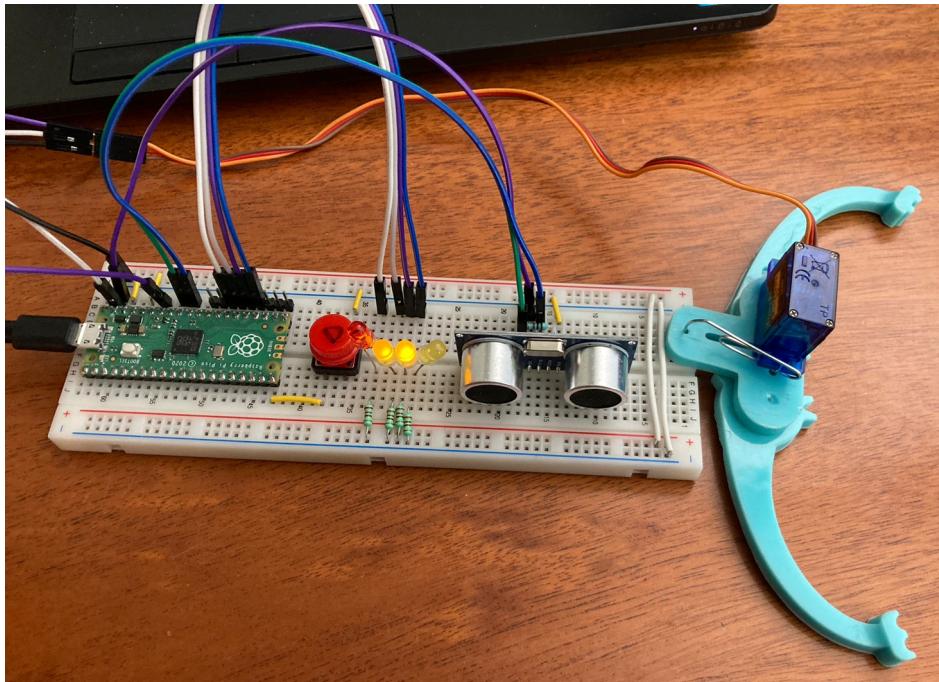
    # lee y muestra la distancia
    lectura = read()
    print("El objeto se encuentra a una distancia de ", lectura, " cm")

    # checa si está demasiado cerca y cierra
    if lectura < 10:
        pwmServo.duty_u16(cerrado)
        cuenta += 1
        alarma = 1
        ante = 1

    utime.sleep(0.1)

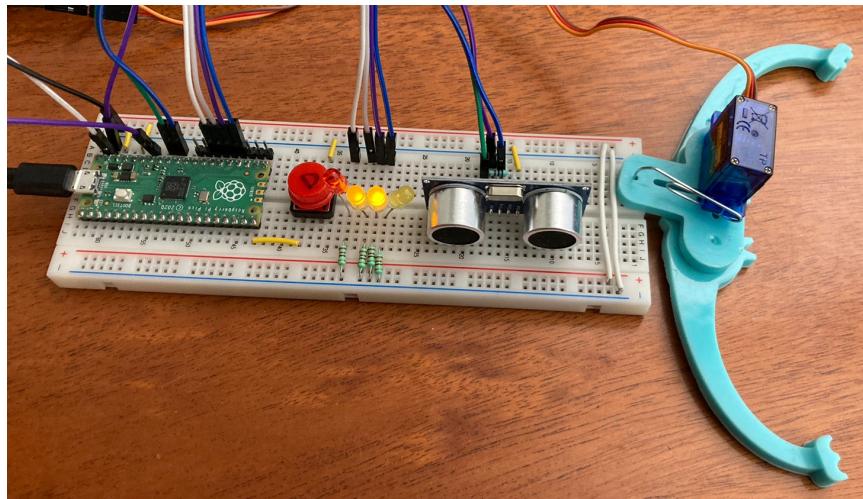
#####
###                                     ###
##          FIN DEL PROGRAMA DEL PRIMER NÚCLEO          ###
##                                     ###
#####
```

## Ensamblado del circuito

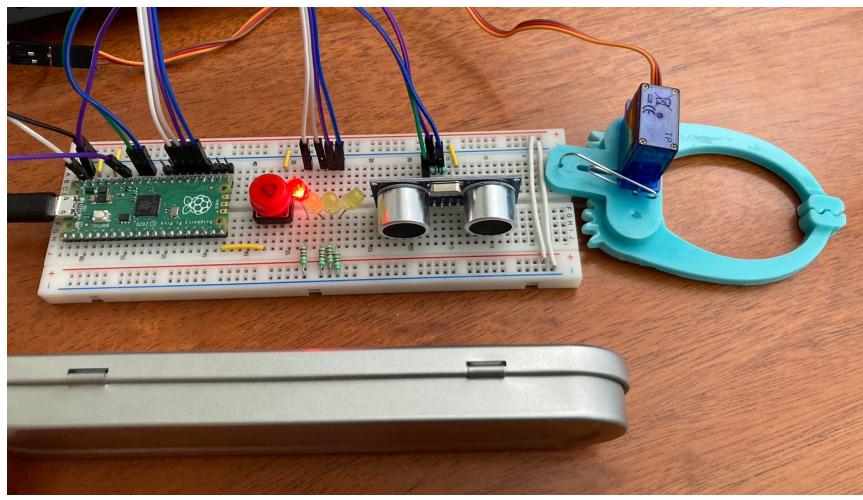


## Evidencias de la ejecución

Caso en el que él no hay algo cerca del sensor: las pinzas continúan abiertas y los leds encendidos son los amarillos.



Caso en el que hay un objeto cerca del sensor: se cierran las pinzas y se enciende el led rojo.



### Mensajes de la computadora:

Cuando se cierra por primera vez por detección de un objeto cercano.

```
COM12 - PuTTY

El objeto se encuentra a una distancia de 56.0 cm
El objeto se encuentra a una distancia de 56.0 cm
El objeto se encuentra a una distancia de 56.0 cm
El objeto se encuentra a una distancia de 56.0 cm
El objeto se encuentra a una distancia de 61.0 cm
El objeto se encuentra a una distancia de 60.0 cm
El objeto se encuentra a una distancia de 59.0 cm
El objeto se encuentra a una distancia de 58.0 cm
El objeto se encuentra a una distancia de 58.0 cm
El objeto se encuentra a una distancia de 57.0 cm
El objeto se encuentra a una distancia de 56.0 cm
El objeto se encuentra a una distancia de 55.0 cm
El objeto se encuentra a una distancia de 55.0 cm
El objeto se encuentra a una distancia de 54.0 cm
El objeto se encuentra a una distancia de 57.0 cm
El objeto se encuentra a una distancia de 16.0 cm
El objeto se encuentra a una distancia de 64.0 cm
El objeto se encuentra a una distancia de 58.0 cm
El objeto se encuentra a una distancia de 6.0 cm

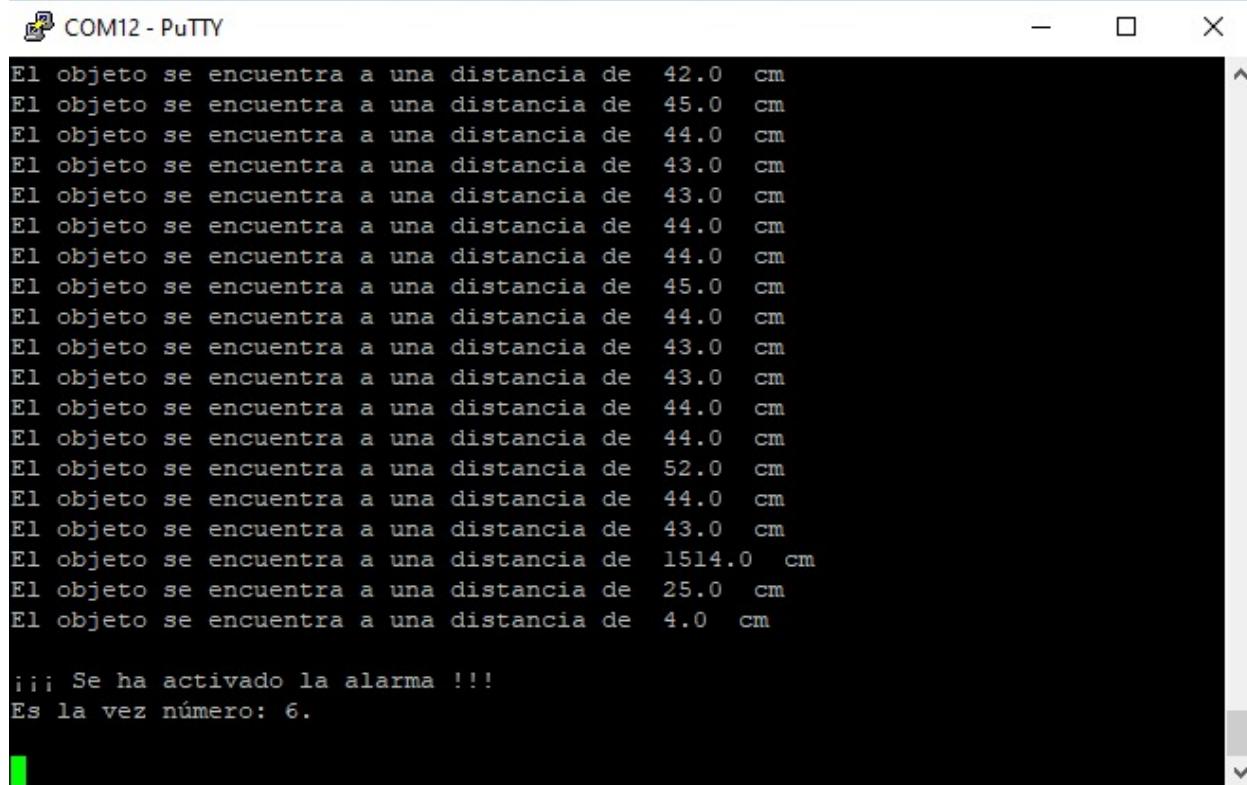
;;; Se ha activado la alarma !!!
Es la vez número: 1.
```

Cuando funciona constantemente sin detectar un objeto demasiado cerca.

```
COM12 - PuTTY

El objeto se encuentra a una distancia de 47.0 cm
El objeto se encuentra a una distancia de 1515.0 cm
El objeto se encuentra a una distancia de 50.0 cm
El objeto se encuentra a una distancia de 45.0 cm
El objeto se encuentra a una distancia de 46.0 cm
El objeto se encuentra a una distancia de 43.0 cm
El objeto se encuentra a una distancia de 42.0 cm
El objeto se encuentra a una distancia de 39.0 cm
El objeto se encuentra a una distancia de 39.0 cm
El objeto se encuentra a una distancia de 39.0 cm
El objeto se encuentra a una distancia de 40.0 cm
El objeto se encuentra a una distancia de 41.0 cm
El objeto se encuentra a una distancia de 42.0 cm
El objeto se encuentra a una distancia de 39.0 cm
El objeto se encuentra a una distancia de 39.0 cm
El objeto se encuentra a una distancia de 39.0 cm
El objeto se encuentra a una distancia de 38.0 cm
El objeto se encuentra a una distancia de 40.0 cm
El objeto se encuentra a una distancia de 40.0 cm
El objeto se encuentra a una distancia de 39.0 cm
El objeto se encuentra a una distancia de 1515.0 cm
El objeto se encuentra a una distancia de 39.0 cm
El objeto se encuentra a una distancia de 38.0 cm
```

Cuando después de muchos cierres, por sexta vez se vuelve a cerrar al detectar un objeto demasiado cercano.



The screenshot shows a terminal window titled "COM12 - PuTTY". The window contains a black background with white text. The text displays a series of messages from a device, starting with repeated distance measurements (42.0 cm, 45.0 cm, 44.0 cm, 43.0 cm, 43.0 cm, 44.0 cm, 44.0 cm, 45.0 cm, 44.0 cm, 43.0 cm, 43.0 cm, 44.0 cm, 44.0 cm, 52.0 cm, 44.0 cm, 43.0 cm, 1514.0 cm, 25.0 cm, 4.0 cm). Following these, there is a message indicating an alarm has been activated (";;; Se ha activado la alarma !!!") and the count of detections ("Es la vez número: 6."). The PuTTY interface includes standard window controls (minimize, maximize, close) and scroll bars on the right side.

```
El objeto se encuentra a una distancia de 42.0 cm
El objeto se encuentra a una distancia de 45.0 cm
El objeto se encuentra a una distancia de 44.0 cm
El objeto se encuentra a una distancia de 43.0 cm
El objeto se encuentra a una distancia de 43.0 cm
El objeto se encuentra a una distancia de 44.0 cm
El objeto se encuentra a una distancia de 44.0 cm
El objeto se encuentra a una distancia de 45.0 cm
El objeto se encuentra a una distancia de 44.0 cm
El objeto se encuentra a una distancia de 43.0 cm
El objeto se encuentra a una distancia de 43.0 cm
El objeto se encuentra a una distancia de 44.0 cm
El objeto se encuentra a una distancia de 44.0 cm
El objeto se encuentra a una distancia de 52.0 cm
El objeto se encuentra a una distancia de 44.0 cm
El objeto se encuentra a una distancia de 43.0 cm
El objeto se encuentra a una distancia de 1514.0 cm
El objeto se encuentra a una distancia de 25.0 cm
El objeto se encuentra a una distancia de 4.0 cm

;;; Se ha activado la alarma !!!
Es la vez número: 6.
```

## Conclusiones de la elaboración del Proyecto Final

### Carolina Álvarez Rodea

En este proyecto se utilizaron conocimientos de los proyectos anteriores, así como los aprendidos durante el curso. Se buscó implementar un sistema de aplicación real (ya sea el procesamiento de señales o el sistema de seguridad) a través del microcontrolador Raspberry Pi Pico, con esto, una vez más comprobamos los alcances que puede tener este microcontrolador que aunque es pequeño tiene una gran capacidad de procesamiento. Para la realización de los dos proyectos fue necesario configurar los pines de entrada y salida de nuestro microcontrolador según los componentes que utilizamos, también fue necesario investigar sobre los protocolos de comunicación como el SPI, USB y UART, y por supuesto buscar las bibliotecas que nos sirvieran para facilitar el desarrollo del código.

Con todo lo realizado en el proyecto considero que se cumplieron los objetivos del mismo, concluyendo que el microcontrolador tiene mucho potencial y que es fácil de manejar debido a que se puede codificar en un lenguaje de alto nivel que es más comprensible para los programadores.

### Dulce Elizabeth Mendoza de la Vega

A lo largo del semestre logramos analizar e implementar distintos conceptos y virtudes de la tarjeta Raspberry Pi Pico la cual nos permitió cargar cada uno de los proyectos realizados en lenguaje micropython, además de apoyarnos con un circuito el cual nos permitió observar el objetivo de dichos proyectos. Ahora bien, durante este proyecto logramos hacer uso de los protocolos UART, SPI y USB, específicamente para el primer proyecto logramos comprender y aplicar los conceptos explicados por el profesor, mientras que el segundo proyecto nos permitió investigar y plantearnos una situación o problemática de la vida real, partiendo de algo sencillo como lo fue el registro de un sensor logramos desarrollar un simulador de una alerta de seguridad. Con esto solo podemos concluir que logramos comprender los conceptos a tal grado de poder ser capaces de desarrollar una herramienta de la vida real.



### Juan Manuel Nava Rosales

Con el desarrollo de los proyectos realizados durante el curso y los temas vistos en clase nos proporcionó las bases para la implementación de aplicaciones matemáticas y con ámbito didáctico, hicimos uso del microcontrolador Raspberry Pi Pico en complemento con módulos (sensores, bus SPI, bus de transmisión serial asíncrona, bus serial universal, entre otros) que por su capacidad y manejo con el lenguaje Python, nos fue de mucha utilidad para el curso. Los objetivos se han cumplido de la mejor manera, al desarrollar en primera instancia el manejo de ondas (generación, lectura, transmisión) partiendo desde las operaciones en el microcontrolador y finalizando con el control de las mismas en software, e incluso operaciones de detección de magnitudes físicas para encontrar alguna variación peligrosa o en situación considerablemente preocupante de aplicaciones reales como en sistemas de seguridad, entre otras aplicaciones que son de suma importancia en el cuidado de la salud, sin embargo, existe un amplio panorama de aplicaciones en distintos sectores como se vió a lo largo del curso. El conocimiento impartido me pareció bastante claro y conciso, de mucha utilidad el desarrollo de los últimos proyectos.

### Iñaky Ordiales Caballero

Durante la elaboración de este proyecto realmente pudimos demostrar los conocimientos aprendidos a lo largo del curso. Desde la codificación de un microcontrolador y la interacción de sus pines como entradas y salidas, hasta la comunicación a través de protocolos como UART, SPI y USB. Creo que se logra cumplir con los objetivos y además podemos apreciar todas las posibilidades que nos ofrecen los microcontroladores y las microcomputadoras. Es muy interesante la actualidad de la tecnología en donde cada vez este tipo de tarjetas pequeñas tienen más características y habilidades, lo que nos permite utilizarlas en muchos ámbitos diferentes. Ya sea como filtros digitales de señales, o como sistemas de seguridad, las microcomputadoras no sólo son el futuro sino también el presente de la computación. Me gustó realizar el proyecto porque pudimos empezar a adentrarnos a aplicaciones y sistemas reales que se pueden implementar formalmente en un futuro.

## Referencias

- Raspberry Pi Ltd. (2021). *Raspberry Pi Documentation - MicroPython*. Raspberry Pi. <https://www.raspberrypi.com/documentation/microcontrollers/micropython.html>
- Crossing, K., [KipCrossing]. (s. f.). *GitHub - KipCrossing/Micropython-AD9833: This script is written in python 3.x for interfacing the AD9833 with microcontrollers with micropython (specifically the PyBoard) over SPI.* GitHub. <https://github.com/KipCrossing/Micropython-AD9833>
- Analog Devices. (2019). *Data Sheet AD9833. Low Power, 12.65mW, 2.3 V to 5.5 V, Programmable Waveform Generator*. Recuperado 15 de enero de 2023, de <https://www.analog.com/media/en/technical-documentation/data-sheets/AD9833.pdf>
- colaboradores de Wikipedia. (2022, 13 julio). *Circuito RC*. Wikipedia, la enciclopedia libre. [https://es.wikipedia.org/wiki/Circuito\\_RC](https://es.wikipedia.org/wiki/Circuito_RC)
- *Todo lo que necesitas saber sobre Filtros RC.* (s. f.). <https://solectroshop.com/es/blog/todo-lo-que-necesitas-saber-sobre-filtros-rc-n52>
- Camarillo, A. (2021, 29 julio). *Tutorial #5 de Raspberry Pi Pico: el servomotor.* 330ohms. <https://blog.330ohms.com/2021/02/12/tutorial-5-de-raspberry-pi-pico-el-servomotor/>
- 330Ohms. (2021, 28 diciembre). *Tutorial #8 de Raspberry Pi Pico: sensor ultrasónico.* 330ohms. Recuperado 15 de enero de 2023, de <https://blog.330ohms.com/2021/12/28/tutorial-8-de-raspberry-pi-pico-sensor-ultrasonico/>