

**Universidad
del Caribe**

2000

CANCUN, QUINTANA ROO, MEXICO

CONOCIMIENTO Y CULTURA PARA EL DESARROLLO HUMANO

Ingeniería en Datos e Inteligencia Organizacional

Rastreador GPS

Parcial 3

Integrantes:

Aznar Hernández Andrea Zazil
Cuevas Alamo Evali
Dzib de la Rosa Adriana
López Ramírez Diego Baudel
Monrreal Pool Ricardo
Montiel Uicab Gabriela Carolyn
Ramayo Aké Cynthia Silvana

Profesor:

Ismael Jiménez Sánchez

Materia:

Organización y Diseño de Computadoras

Introducción

Este proyecto es con la finalidad de presentar nuestro proyecto final para la materia de Organización y Diseño de Computadoras, el cual debe ser un proyecto con el uso de alguna microcomputadora tales como:

- Raspberry Pi
- Arduino
- Banana Pi
- Orange Pi
- Etcétera...

Este proyecto está enfocado principalmente en un Rastreador GPS que permite, mediante un módulo gps, rastrear la localización de esté mismo, mientras es llevado cargado, puede ir registrando el recorrido hecho por alguien, además de que una vez los datos han sido guardados (latitud, longitud y hora), los datos son guardados en un archivo CSV para luego ser importados a un maps en Google Maps y tener una representación visual del recorrido hecho.

Rastreador GPS

El Rastreador GPS consiste en la recepción de datos satelitales de ubicación a través de la constelación de satélites GPS, los cuales son recibidos a un módulo NEO-6M GPS, con datos mandados a una Raspberry Pi Pico H, para finalizar el ciclo mandando esos datos a una pantalla OLED de 0,96" SSD1306 y observar 4 cosas esenciales:

1. Número de satélites que estamos recibiendo datos
2. Latitud actual
3. Longitud actual
4. Horario actual.

Tanto latitud, longitud y horario son guardados en un archivo dentro de la memoria de la Raspberry Pi Pico.

Datos esenciales a tomar en cuenta en este proyecto es que el módulo GPS NEO-6M requiere de mínimo la recepción de 4 satélites, ya que 1

satélite debe proporcionar latitud, 1 más, longitud, 1 más debe proporcionar altura y 1 último proporciona la referencia de horario. El módulo NEO-6M hace una comunicación en serie con la Raspberry Pi Pico mandando los datos mediante NMEA 0183, el cual es una norma estándar industrial de aplicación que define las condiciones que se deben respetar las señales eléctricas en la transmisión de una señal entre dispositivos de navegación.

Ejemplo:

SGPGGA, 092750, 000, 5321.6802, N, 00630.3372, W, 1, 8, 1, 03, 61, 7, M, 55, 2, M,, *76

La sintaxis es que está separado por comas cada sentencia.

\$GPGGA: Hace referencia al dispositivo de proveniencia, en este caso GGA (datos de posición del sistema de posicionamiento global)

092750.000: Hora (09:27:50)

5321.6802. N: Latitud 53. grados 21.6802' N

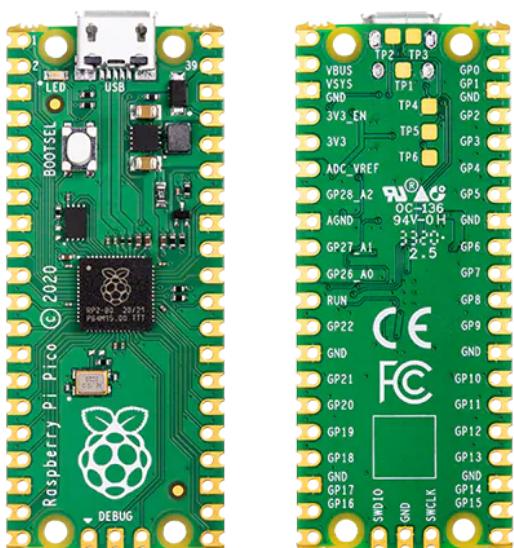
00630.3372. W: Longitud 6. grados 30.3372'W

08· Cantidad de satélites

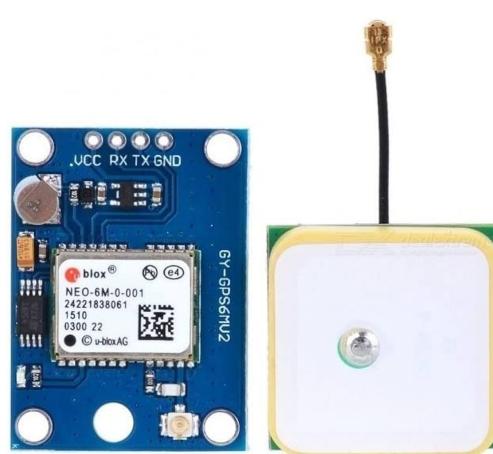
61.7: Nivel sobre el mar, en metros

Materiales

1. Raspberry Pi Pico



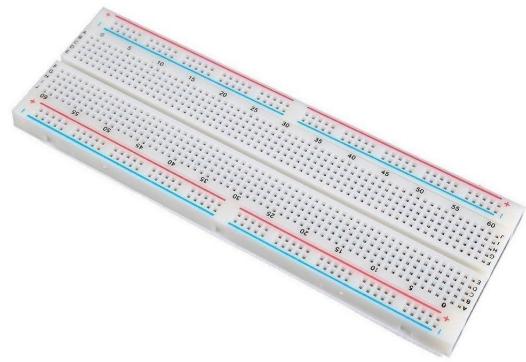
2. Módulo GPS NEO-6M



3. Pantalla OLED SSD1306



4. Protoboard



5. Jumpers



6. Baterías AA con portapilas



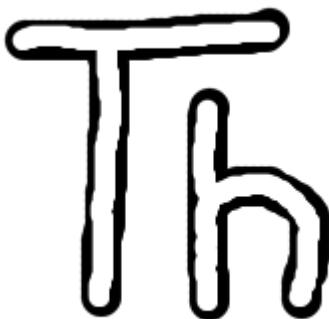
7. Cautín



8. Estaño para soldar



8. Computadora con Thonny



9. Librerías (Software)

```
micropython-ssd1306 / ssd1306.py
```

lurch · Update ssd1306.py to latest version from upstream micropython repo · d1fca2a · 3 years ago · History

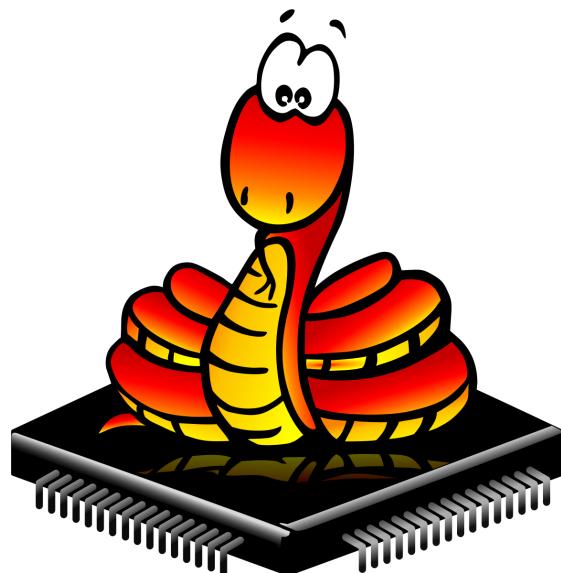
Code Blame 155 lines (136 loc) · 4.48 kB · Code 55% faster with GitHub Copilot

```
1 # MicroPython SSD1306 (I2C) driver, I2C and SPI interfaces
2
3 from microcontroller import const
4 import framebuffer
5
6
7 # register definitions
8 SET CONTRAST = const(0x0d)
9 SET_BIAS_T_0N = const(0x04)
10 SET_WKUP_SW = const(0x04)
11 SET_DISP = const(0x02)
12 SET_MADCTL = const(0x20)
13 SET_COLADDR = const(0x02)
14 SET_PAGEADDR = const(0x02)
15 SET_DISP_STARTLINE = const(0x00)
16 SET_SBS_BEWP = const(0x00)
17 SET_NOR_RATIO = const(0x04)
18 SET_OPMODE = const(0x00)
19 SET_DISP_OFFSET = const(0x00)
```

10. Cable tipo micro usb



11. Código en MicroPython



Fuente: Computadoras y Sensores.

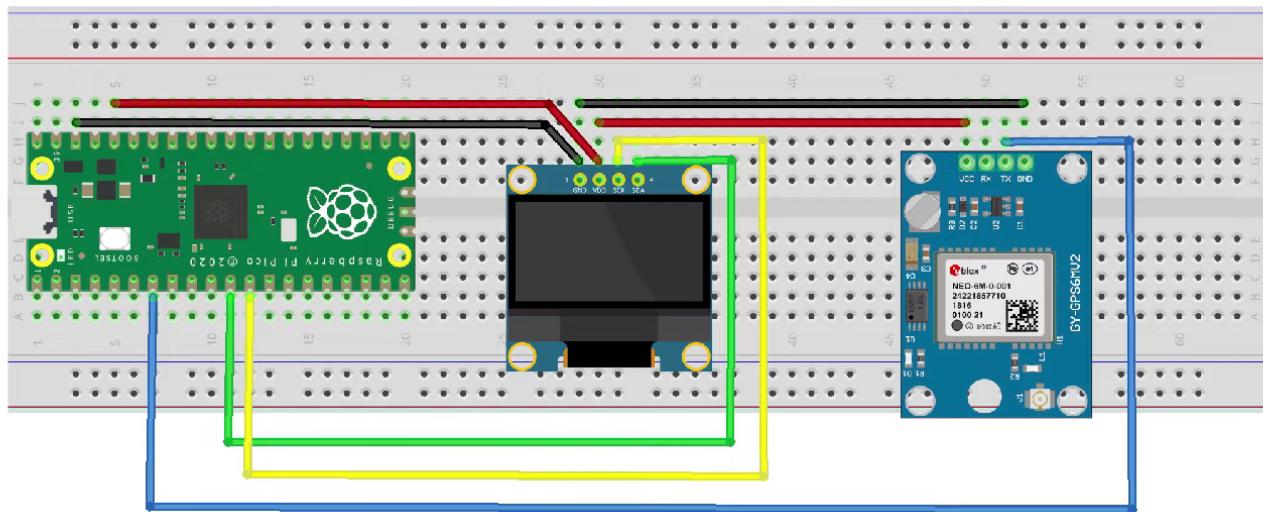
11. Fila de pines



12. Caja decorativa

Conexionado

General



Respecto a la Raspberry Pi Pico

Pin **36**: Para alimentación a módulo OLED en **VDD**.

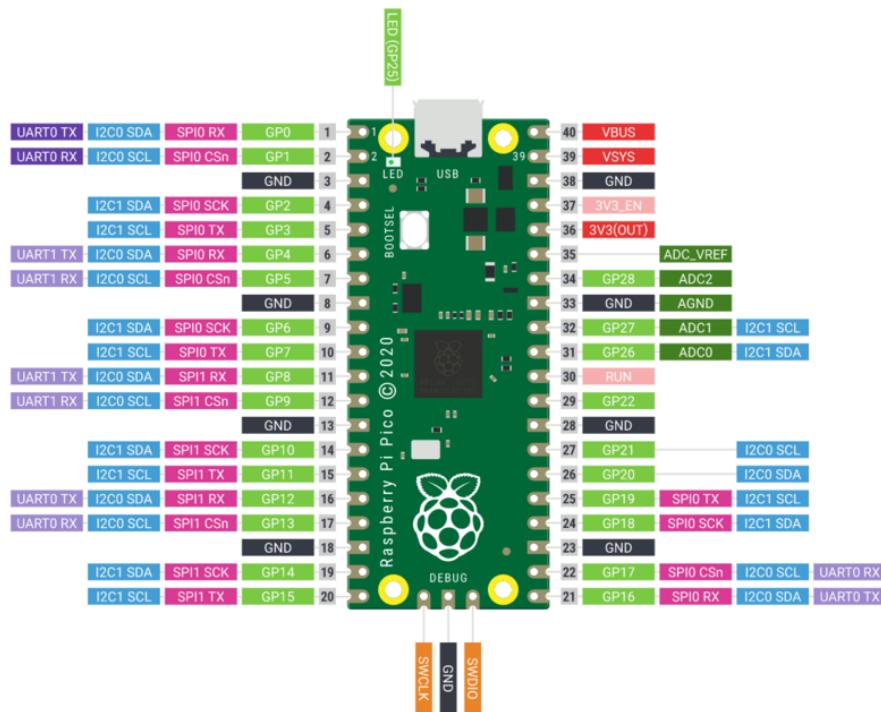
Pin **38**: Para alimentación a módulo OLED en **GND**.

Pin **7**: Conectado a pin **TX** del módulo GPS.

Pin **11**: Conectado a pin **SDA** (Serial Data) al módulo OLED.

Pin **12**: Conectado a pin **SCK** (Serial Clock) al módulo OLED.

Pinout:



Nota: Se agregaron 2 pines más en cadena a la Pico, en conexión de los pines **36** y **38** para las baterías.

Respecto al módulo OLED SSD1306

Pin **VDD**: En cadena a **VCC** del módulo GPS.

Pin **GND**: En cadena a **GND** del módulo GPS.

Proceso de creación

Respecto a Hardware:

1. Se hizo el soldado de pines en la Raspberry Pi Pico y el módulo GPS, afortunadamente la pantalla OLED venía soldada.
2. Se hizo el debido ensamblado de cada componente en la protoboard, tanto Raspberry Pi Pico, pantalla OLED y módulo GPS.
3. Se hizo el conexionado de cada componente de acuerdo a los esquemas de conexionado y especificaciones.
4. Se elaboró la conexión del portabaterías con jumpers machos para conexión en la protoboard.
5. Decoración y fijación de todos los componentes.

Respecto a Software:

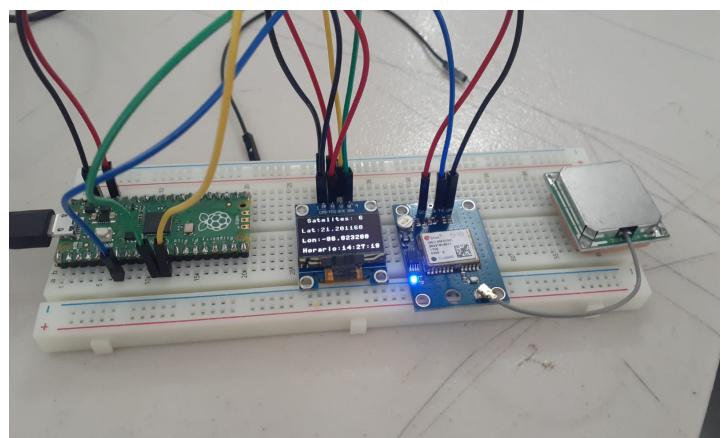
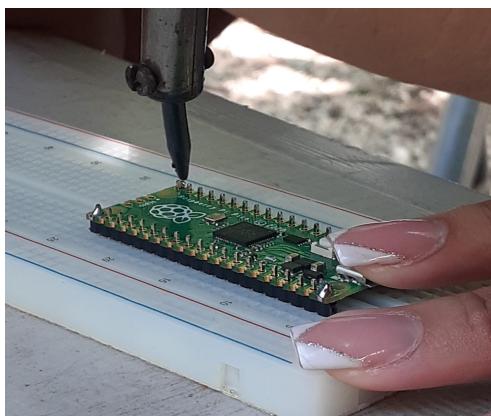
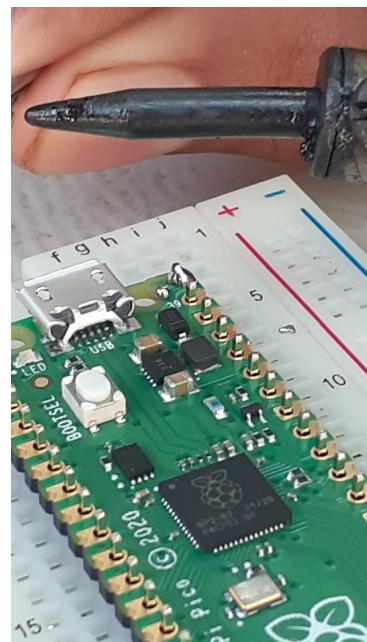
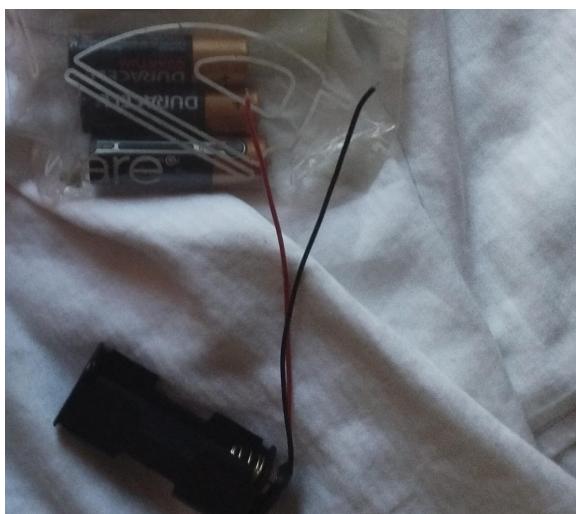
1. Descarga del programa Thonny que hace uso de Micropython como lenguaje de programación.
2. Se hizo la debida instalación y actualización de la Raspberry Pi Pico, para que pudiese configurarse a placer con el compilador Thonny.
3. Se hizo la descarga de librerías para la pantalla OLED SSD1306 y Módulo NEO-6M, tanto en Thonny como dentro de la Raspberry Pi Pico
 - a. Link de librería OLED SSD1306:
<https://github.com/stlehmann/micropython-ssd1306>
 - b. Link de librería Módulo NEO-6M:
<https://github.com/inmcm/micropyGPS>
4. Se agrega código completo y central para el funcionamiento del Rastreador GPS

a. Link del código completo, elaborado por Computadoras y Sensores:

<https://github.com/ComputadorasySensores/Capitulo37>

5. Se modificó el valor de zona horaria a UTC-5.
6. Se modificaron los valores de intervalos, de 30 segundos a 10 segundos para el guardado de datos.

Imágenes de elaboración y proceso del Rastreador GPS



Código usado para el proyecto

```
from machine import Pin, UART, I2C
    # se importan pines UART e I2C (Módulo GPS y Pantalla Oled
    respectivamente)
import utime, time
    # se importan valores horarios
from ssd1306 import SSD1306_I2C
from micropyGPS import MicropyGPS
    # se importan ambas librerías anteriormente descargadas

i2c = I2C(0, sda=Pin(8), scl=Pin(9), freq=400000)
    # se establece el objeto i2c, otorgando valores al pin 8 como SDA y
    al pin 9 como SCL, a frecuencia de 40KHz
oled = SSD1306_I2C(128, 64, i2c)
    # se establece el objeto i2c, que se otorga la resolución de pantalla
    de 128 por 64 pixeles

modulo_gps = UART(1, baudrate=9600, tx=Pin(4),
rx=Pin(5))
    # se establece el objeto modulo_gps, para representar la conexión
    serie por UART, que recibe información del módulo GPS,

Zona_Horaria = -5
gps = MicropyGPS(Zona_Horaria)
    # se establece la zona horaria, para luego pasar ese valor al objeto
    "gps"

def convertir(secciones):
    # secciones[0] contiene los grados
    if (secciones[0] == 0):
        return None
    # secciones[1] contiene los minutos
    data = secciones[0]+(secciones[1]/60.0)
    # secciones[2] contiene 'E', 'W', 'N', 'S'
    if (secciones[2] == 'S'):
```

```

        data = -data
    if (secciones[2] == 'W'):
        data = -data

    data = '{0:.6f}'.format(data) # 6 digitos decimales
return str(data)

ultimo_valor = 0
intervalo_valores = 10
# dos variables con segundos, empezando desde 0, para cada 10 segundos
guardar datos

while True:
    largo = modulo_gps.any()
    # guarda los valores del mensaje enviado por el módulo gps en "largo"
    if largo > 0:
        b = modulo_gps.read(largo)
        # se manda a "b" los datos en formato NMEA
        for x in b:
            msg = gps.update(chr(x))
            # se manda y se hace el bucle por carácter a la vez, la
            # función gps.update se encuentra en la biblioteca y hace
            # una validación de datos
    latitud = convertir(gps.latitude)
    longitud = convertir(gps.longitude)
    # convierte ambos valores en un formato string, adecuados para
    # mostrarse en la pantalla

    if (latitud == None or longitud == None):
        oled.fill(0)
        oled.text("Datos no", 35, 25)
        oled.text("disponibles", 22, 40)
        oled.show()
        continue

    # esta condición sirve en caso de que latitud o longitud no tengan
    # valores, y se muestre un mensaje de que no hay datos

```

```

t = gps.timestamp
# timestamp es un formato de horario
#t[0] => horas : t[1] => minutos : t[2] => segundos
horario = '{:02d}:{:02d}:{:02d}'.format(t[0],t[1],
t[2])

oled.fill(0)
oled.text('Satelites: ' +
str(gps.satellites_in_use), 10, 0)
oled.text('Lat:' + latitud, 0, 18)
oled.text('Lon:' + longitud, 0, 36)
oled.text('Horario:' + horario, 0, 54)
oled.show()

# para mostrar todo lo necesario en la pantalla OLED, número de
satélites conectados, latitud, longitud y horario

if (time.time() - ultimo_valor) >
intervalo_valores:
    # se resta el tiempo actual menos el último_valor (0) y si es mayor
    que intervalo_valores (10), prosigue el if
    archivo = open("datos_gps.csv", "a")
    archivo.write(latitud + ",")
    archivo.write(longitud + ",")
    archivo.write(horario + "\n")
    archivo.close()
    ultimo_valor = time.time()
    # aquí dentro se escribe dentro del archivo csv cada dato,
    desde latitud y longitud y horario

```

Costos	
Raspberry Pi Pico H	\$120
Pantalla OLED SSD1306	\$110
Módulo GPS NEO-6M	\$110
Jumpers	\$70
Envío	\$104
Estaño	\$20
Tira de 38 pines	\$20
TOTAL:	\$554