

```
#Tx solo para nRF24L01 y oled -prueba
conexión exitosa from machine import Pin,
SPI
import utime
from nrf24l01 import NRF24L01

# SPI0 alterno
spi = SPI(0, sck=Pin(6), mosi=Pin(7),
miso=Pin(4)) csn = Pin(15, Pin.OUT,
value=1)
ce = Pin(14, Pin.OUT, value=0)

# Canal 40, payload 16
nrf = NRF24L01(spi, csn, ce, 40, 16)

TX_ADDR = b'\xE1\xF0\xF0\xF0\xF0'
RX_ADDR =
b'\xD2\xF0\xF0\xF0\xF0'
nrf.open_tx_pipe(TX_ADDR)
nrf.open_rx_pipe(1, RX_ADDR)
# ---- Ajustes robustos sin ACK ----
# Sin auto-ack y sin retransmisiones
nrf.reg_write(0x01, 0x00) # EN_AA = 0
nrf.reg_write(0x04, 0x00) # SETUP_RETR = 0
# 250 kbps y -18 dBm
rf = nrf.reg_read(0x06) & 0b11100011
rf |= (1<<6) | (0<<4) | (0<<2) # DR_LOW=1, DR_HIGH=0,
RF_PWR=00 nrf.reg_write(0x06, rf)
# Limpiar flags
nrf.reg_write(0x07, 0x70)

nrf.stop_listening()

print("TX listo (canal 40, 250kbps, sin ACK, -18dBm).")
contador = 0

while True:
```

```
# Mensaje fijo de 16 bytes (relleno con espacios si
# hace falta) msg = ("HELLO %04d" % (contador %
10000)).encode()
msg = (msg + b" " * 16)[:16]

try:
    # Limpiar flags antes de cada envío por si quedó
    MAX_RT nrf.reg_write(0x07, 0x70)
    nrf.send(msg)
    print("☒ Enviado:", msg)
except OSError:
    # En caso de fallo, limpiar y vaciar FIFO TX
    print(" Fallo de envío")
    nrf.reg_write(0x07, 0x70)
    # FLUSH_TX (comando 0xE1): algunos drivers exponen método,
    otros no. try:
        nrf.flush_tx()
    except:
        pass

    contador += 1
    utime.sleep_ms(300) # más tiempo entre envíos
```

```

<sin nombre> [TX.py] 
1 from machine import Pin, SPI
2 import utime
3 from nrf24l01 import NRF24L01
4
5 # SPI0 alterno
6 spi = SPI(0, sck=Pin(6), mosi=Pin(7), miso=Pin(4))
7 csn = Pin(15, Pin.OUT, value=1)
8 ce = Pin(14, Pin.OUT, value=0)
9
10 # Canal 40, payload 16
11 nrf = NRF24L01(spi, csn, ce, 40, 16)
12
13 TX_ADDR = b'\xE1\xF0\xF0\xF0\xF0'
14 RX_ADDR = b'\xD2\xF0\xF0\xF0\xF0'
15 nrf.open_tx_pipe(TX_ADDR)
16 nrf.open_rx_pipe(1, RX_ADDR)
17
18 # ---- Ajustes robustos sin ACK ----
19 # Sin auto-ack y sin retransmisiones
20 nrf.reg_write(0x01, 0x00)          # EN_AA = 0
21 nrf.reg_write(0xA0, 0x00)          # CPT0D_RTR = 0

```

Concilia >

```

MPY: soft reboot
TX listo (canal 40, 250kbps, sin ACK, -18dBm).
✓ Enviado: b'HELLO 0000 '
✓ Enviado: b'HELLO 0001 '
✓ Enviado: b'HELLO 0002 '
✓ Enviado: b'HELLO 0003 '
✓ Enviado: b'HELLO 0004 '

```

#Rx solo para nRF24L01 y oled -prueba
conexión exitosa

```
from machine import Pin, SPI,
I2C
import utime
from nrf24l01 import NRF24L01
```

```
# OLED I2C1 GP11/GP10 (addr 0x3d según tu
escaneo) try:
```

```
from ssd1306 import SSD1306_I2C
i2c = I2C(1, scl=Pin(11), sda=Pin(10),
freq=400000) oled = SSD1306_I2C(128,
64, i2c, addr=0x3D)
except:
oled = None
```

```
# Radio SPI0 alterno: SCK=GP6, MOSI=GP7,
MISO=GP4 spi = SPI(0, sck=Pin(6),
mosi=Pin(7), miso=Pin(4))
csn = Pin(15, Pin.OUT, value=1)
```

```

ce = Pin(14, Pin.OUT, value=0)

# Canal 40, payload 16 bytes
nrf = NRF24L01(spi, csn, ce, 40, 16)

# Direcciones
TX_ADDR = b'\xE1\xF0\xF0\xF0\xF0'
RX_ADDR = b'\xD2\xF0\xF0\xF0\xF0'
nrf.open_tx_pipe(TX_ADDR)
nrf.open_rx_pipe(1, RX_ADDR)

# ---- Ajustes robustos sin ACK ----
# EN_AA (0x01) = 0x00 -> sin auto-ack en
# todos los pipes nrf.reg_write(0x01, 0x00)
# SETUP_RETR (0x04) = 0x00 -> sin
# retransmisiones nrf.reg_write(0x04,
# 0x00)
# RF_SETUP (0x06):
# DR_LOW=1 (250kbps), DR_HIGH=0
# RF_PWR=00 (-18 dBm)
rf = nrf.reg_read(0x06) & 0b11100011
rf |= (1<<6) | (0<<4) | (0<<2)
nrf.reg_write(0x06, rf)
# Limpiar flags
nrf.reg_write(0x07, 0x70)

nrf.start_listening()

print("RX listo (canal 40, 250kbps, sin ACK, -18dBm).")

while True:
    # Leer TODO lo que haya en FIFO
    while nrf.any():
        raw = nrf.recv()
        try:
            txt = raw.decode().strip('\x00')
        except:
            txt = ''.join(chr(b) for b in raw if 32 <= b < 127)

        print("PKT:", txt)
        if oled:
            oled.fill(0)

```

```
oled.text("NRF24 RX OK", 0, 0)
oled.text("Ch40 250k noACK", 0,
12) oled.text(txt[:16], 0, 32)
oled.show()
```

```
utime.sleep_ms(5)
```

[RX.py]

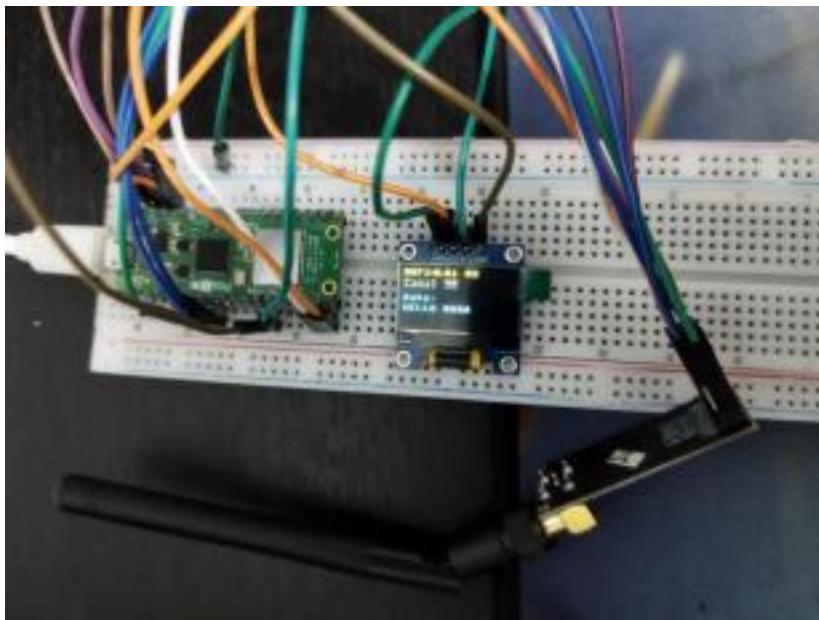
```
1 from machine import Pin, SPI, I2C
2 import utime
3 from nrf24l01 import NRF24L01
4
5 # OLED I2C1 GP11/GP10 (addr 0x3d según tu escaneo)
6 try:
7     from ssd1306 import SSD1306_I2C
8     i2c = I2C(1, scl=Pin(11), sda=Pin(10), freq=400000)
9     oled = SSD1306_I2C(128, 64, i2c, addr=0x3D)
10 except:
11     oled = None
12
13 # Radio SPI0 alternos: SCK=GP6, MOSI=GP7, MISO=GP4
14 spi = SPI(0, sck=Pin(6), mosi=Pin(7), miso=Pin(4))
15 csm = Pin(15, Pin.OUT, value=1)
16 ce = Pin(14, Pin.OUT, value=0)
17
18 # Canal 40, payload 16 bytes
19 nrf = NRF24L01(spi, csm, ce, 40, 16)
20
21 # Direcciones
22 TX_ADDR = b'\xE1\xF0\xF0\xF0\xF0'
23 RX_ADDR = b'\xD2\xF0\xF0\xF0\xF0'
24 nrf.open_tx_pipe(TX_ADDR)
25 nrf.open_rx_pipe(1, RX_ADDR)
26
27 # ---- Ajustes robustos sin ACK ----
28 # EN_AA (0x01) = 0x00 -> sin auto-ack en todos los pipes
29 nrf.reg_write(0x01, 0x00)
```

C

Consola

```
NPY: soft reboot
RX listo (canal 40, 250kbps, sin ACK, -18dBm).
Pkt: HELLO 0001
Pkt: HELLO 0004
Pkt: HELLO 0015
Pkt: HELLO 0019
Pkt: HELLO 0023
Pkt: HELLO 0027
```

CONEXIÓN EXITOSA



#Código para probar la oled rx solamente:

```
from machine import Pin, I2C  
from ssd1306 import SSD1306_I2C  
  
# crea el objeto I2C con la dirección 0x3C que  
detectamos i2c = I2C(1, scl=Pin(11),  
sda=Pin(10), freq=400000) oled =  
SSD1306_I2C(128, 64, i2c, addr=0x3C)
```

```
oled.fill(0)  
oled.text("OLED OK", 0, 0)  
oled.text("Direccion 0x3C", 0, 16)  
oled.text("Todo funciona :)", 0, 32)  
oled.show()
```

#código para probar el servo solamente.

```
from machine import Pin, PWM  
import time
```

```
servo = PWM(Pin(15))  
servo.freq(50)  
  
def servo_deg(d):  
    d = max(0, min(180, int(d)))  
    us = 500 + (2000*d)//180  
    servo.duty_ns(us*1000)
```

```

while True:
    for a in range(0,181,30):
        servo_deg(a); time.sleep(0.5)
    for a in range(180,-1,-30):
        servo_deg(a); time.sleep(0.5)

```

Código para probar acelerómetro solamente (MPU-6050)

```

from machine import Pin, I2C
import utime, struct
import math
print(" TEST MPU-6050 - Mostrando datos en
consola") print("=" * 50)

```

---- Configuración I2C ----

```

# SCL = GP17 (pin 22), SDA = GP16 (pin 21)
i2c = I2C(0, scl=Pin(17), sda=Pin(16),
freq=400000) MPU_ADDR = 0x68
# ---- Función para inicializar
MPU-6050 ---- def
inicializar_mpu():
    try:
        # Despertar el MPU-6050
        i2c.writeto_mem(MPU_ADDR, 0x6B,
b'\x00') utime.sleep_ms(100)

```

Configurar rango del giroscopio

(±250°/s)

```

i2c.writeto_mem(MPU_ADDR, 0x1B,
b'\x00')

```

Configurar rango del acelerómetro

(±2g) i2c.writeto_mem(MPU_ADDR,

```

0x1C, b'\x00')

```

```

print("☑ MPU-6050 inicializado
correctamente") return True
except Exception as e:
    print(f"✗ Error inicializando MPU-6050:
{e}") return False

```

---- Función para leer acelerómetro ----

```

def leer_acelerometro():
    try:

```

```

data = i2c.readfrom_mem(MPU_ADDR,
0x3B, 6) acc_x = struct.unpack('>h',
data[0:2])[0] / 16384.0 acc_y =
struct.unpack('>h', data[2:4])[0] / 16384.0
acc_z = struct.unpack('>h', data[4:6])[0] /
16384.0 return acc_x, acc_y, acc_z
except Exception as e:
print(f" ✗ Error leyendo acelerómetro:
{e}") return 0, 0, 0

# ---- Función para leer giroscopio ----
def leer_giroscopo():
try:
data = i2c.readfrom_mem(MPU_ADDR,
0x43, 6) gyro_x = struct.unpack('>h',
data[0:2])[0] / 131.0 gyro_y =
struct.unpack('>h', data[2:4])[0] / 131.0
gyro_z = struct.unpack('>h', data[4:6])[0] /
131.0 return gyro_x, gyro_y, gyro_z
except Exception as e:
print(f" ✗ Error leyendo giroscopio:
{e}") return 0, 0, 0

# ---- Función para leer temperatura ----
def leer_temperatura():
try:
data = i2c.readfrom_mem(MPU_ADDR, 0x41, 2)
temp_raw = struct.unpack('>h', data)[0]
temperatura = (temp_raw / 340.0) + 36.53
return temperatura
except:
return 0

# ---- Función para calcular ángulos de inclinación ----
def calcular_inclinacion(acc_x, acc_y, acc_z):
pitch = math.atan2(acc_x, math.sqrt(acc_y*acc_y + acc_z*acc_z)) * (180 /
math.pi) roll = math.atan2(acc_y, math.sqrt(acc_x*acc_x + acc_z*acc_z)) *
(180 / math.pi) return pitch, roll

# ---- Inicializar MPU-6050 ----
print("Inicializando MPU-6050...")
if not inicializar_mpu():

```

```

print("X No se puede continuar sin MPU-6050")
else:
    print("\n Leyendo datos del MPU-6050...")
    print("Presiona Ctrl+C para detener")
    print("-" * 60)

contador = 0
try:
    while True:
        acc_x, acc_y, acc_z = leer_acelerometro()
        gyro_x, gyro_y, gyro_z = leer_giroscopo()
        temperatura = leer_temperatura()
        pitch, roll = calcular_inclinacion(acc_x, acc_y, acc_z)

        if contador % 10 == 0:
            print("\n" + "=" * 50)
            print(f" LECTURA {contador}")
            print("=" * 50)
            print(" ACELERÓMETRO (g):")
            print(f" X: {acc_x:7.3f} | Y: {acc_y:7.3f} | Z: {acc_z:7.3f}")
            print(" GIROSCOPIO (°/s):")
            print(f" X: {gyro_x:7.1f} | Y: {gyro_y:7.1f} | Z: {gyro_z:7.1f}")
            print(" INCLINACIÓN (°):")
            print(f" Pitch: {pitch:6.1f} | Roll: {roll:6.1f}")
            print(" TEMPERATURA:")
            print(f" {temperatura:5.1f}°C")
            print(" ESTADO:")
            if abs(acc_z - 1.0) < 0.2:
                print(" Sensor en posición horizontal")
            else:
                print(" Sensor inclinado")
            print("-" * 50)

        contador += 1
        utime.sleep_ms(100)
    except KeyboardInterrupt:
        print("\n Test terminado")

```

en Rx se movio el pin CSN del nRF24L01 del GP15 al GP5 pin 7 por que estaba interrumpiendo con Señal servo → GP15 - amarillo que estaba en el mismo.

#Tx con servo y oled

```
from machine import Pin, SPI, ADC
import utime, struct
from nrf24l01 import NRF24L01

# ---- Radio SPI0 (GP6/7/4) ----
spi = SPI(0, sck=Pin(6), mosi=Pin(7), miso=Pin(4))
csn = Pin(15, Pin.OUT, value=1)
ce = Pin(14, Pin.OUT, value=0)

# Canal 40, payload 4 bytes
nrf = NRF24L01(spi, csn, ce, 40, 4)

TX_ADDR = b'\xE1\xF0\xF0\xF0\xF0'
RX_ADDR = b'\xD2\xF0\xF0\xF0\xF0'
nrf.open_tx_pipe(TX_ADDR)
nrf.open_rx_pipe(1, RX_ADDR)

def chk(sync, ang):
    lo = ang & 0xFF
    hi = (ang >> 8) & 0xFF
    return (sync + lo + hi) & 0xFF

SYNC = 0xA5

print("TX joystick -> angulo (canal 40, 250kbps, sin ACK, -18dBm)")
while True:
    ang = leer_angulo()
    paquete = struct.pack("<BHB", SYNC, ang, chk(SYNC, ang)) #
    4 bytes try:
        nrf.reg_write(0x07, 0x70) # limpiar flags antes de cada envío
        nrf.send(paquete)
        print("TX ang:", ang)
    except OSError:
        print("⚠ fallo envio")
        nrf.reg_write(0x07, 0x70)
```

```
try: nrf.flush_tx()
except: pass
utime.sleep_ms(80) # ritmo moderado
```

```
27
28 def chk(sync, ang):
29     lo = ang & 0xFF
30     hi = (ang >> 8) & 0xFF
31     return (sync + lo + hi) & 0xFF
32
33 SYNC = 0xA5
34
35 print("TX joystick -> angulo (canal 40, 250kbps, sin ACK")
36 while True:
37     ang = leer_angulo()
38     paquete = struct.pack("<BHB", SYNC, ang, chk(SYNC,
39     try:
40         nrf.reg_write(0x07, 0x70) # limpiar flags ante
41         nrf.send(paquete)
42         print("TX ang:", ang)
43     except OSError:
44         print("⚠ fallo envio")
45         nrf.reg_write(0x07, 0x70)
46         try: nrf.flush_tx()
47     except: pass
48     utime.sleep_ms(80) # ritmo moderado |
```

oncila >

```
MPY: soft reboot
TX joystick -> angulo (canal 40, 250kbps, sin ACK, -18dBm)
TX ang: 140
TX ang: 140
TX ang: 140
TX ang: 141
TX ang: 140
```

```

>>
38 def chk(sync, ang):
39     lo = ang & 0xFF
40     hi = (ang >> 8) & 0xFF
41     return (sync + [lo + hi]) & 0xFF
42
43 SYNC = 0xA5
44
45 print("TX joystick -> angulo (canal 40, 250kbps, sin ACK, -18dBm")
46 while True:
47     ang = leer_angulo()
48     paquete = struct.pack("<BHB", SYNC, ang, chk(SYNC, ang)) # 4 bytes
49     try:
50         nrf.reg_write(0x07, 0x70) # limpiar flags antes de cada envio
51         nrf.send(paquete)
52         print("TX ang:", ang)
53     except OSError:
54         print("⚠ fallo envio")
55         nrf.reg_write(0x07, 0x70)
56         try: nrf.flush_tx()
57         except: pass
58     utime.sleep_ms(80) # ritmo moderado

```

ensola

```

TX ang: 180
TX ang: 180
TX ang: 180
TX ang: 42
TX ang: 0
TX ang: 0
TX ang: 0
*** ----

```

#Rx con servo y oled

```

from machine import Pin, SPI, I2C, PWM
import utime, struct
from nrf24l01 import NRF24L01
from ssd1306 import SSD1306_I2C

# ----- OLED (dirección confirmada 0x3C) -----
i2c = I2C(1, scl=Pin(11), sda=Pin(10), freq=400000)
oled = SSD1306_I2C(128, 64, i2c, addr=0x3C)

))

# Pines de control
csn = Pin(5, Pin.OUT, value=1) # ← CSN movido a GP5
ce = Pin(14, Pin.OUT, value=0) # CE en GP14

# Inicializa módulo nRF
nrf = NRF24L01(spi, csn, ce, 40, 4) # canal 40, payload 4 bytes

TX_ADDR = b'\xE1\xF0\xF0\xF0\xF0'
RX_ADDR = b'\xD2\xF0\xF0\xF0\xF0'
nrf.open_tx_pipe(TX_ADDR)
nrf.open_rx_pipe(1, RX_ADDR)

```

```
# Configuración robusta: sin auto-ack, 250 kbps, -18 dBm
nrf.reg_write(0x07, 0x70) # limpia flags
nrf.start_listening()

# ----- Verificación de recepción -----
SYNC = 0xA5
def chk(sync, ang):
    """Checksum simple (SYNC + ANG_L + ANG_H)."""
    return (sync + (ang & 0xFF) + ((ang >> 8) & 0xFF)) & 0xFF

ultimo = 90
servo_deg(ultimo)

print("☒ RX servo+OLED listo (ch40, 250kbps, sin ACK, -18dBm,
CSN→GP5)") # ----- Bucle principal -----
# Muestra en OLED
oled.fill(0)
oled.text("NRF24 RX Servo", 0, 0)
oled.text("Ang: %3d deg" % ultimo, 0, 16)
oled.text("Canal 40", 0, 32)
oled.text("Vel 250kbps", 0, 44)
oled.show()
print("RX ang:", ultimo)
utime.sleep_ms(5)
```

```

51 ultimo = 90
52 servo_deg(ultimo)
53
54 print("✓ RX servo+OLED listo (ch40, 250kbps, sin ACK, -18dBm, CSN+GP5)")
55
56 # ----- Bucle principal -----
57 while True:
58     while nrf.any():
59         raw = nrf.recv()    # 4 bytes recibidos
60         try:
61             s, ang, c = struct.unpack("<BHB", raw)
62         except:
63             continue
64
65         ok = (s == SYNC) and (c == chk(s, ang)) and (0 <= ang <= 180)
66         if ok:
67             ultimo = ang
68             servo_deg(ultimo)
69
70             # Muestra en OLED
71             oled.fill(0)
72             oled.text("NRF24 RX Servo", 0, 0)
73             oled.text("Ang: %3d deg" % ultimo, 0, 16)
74             oled.text("Canal 40", 0, 32)
75             oled.text("Vel 250kbps", 0, 44)
76             oled.show()
77             print("RX ang:", ultimo)
78         utime.sleep_ms(5)
79

```

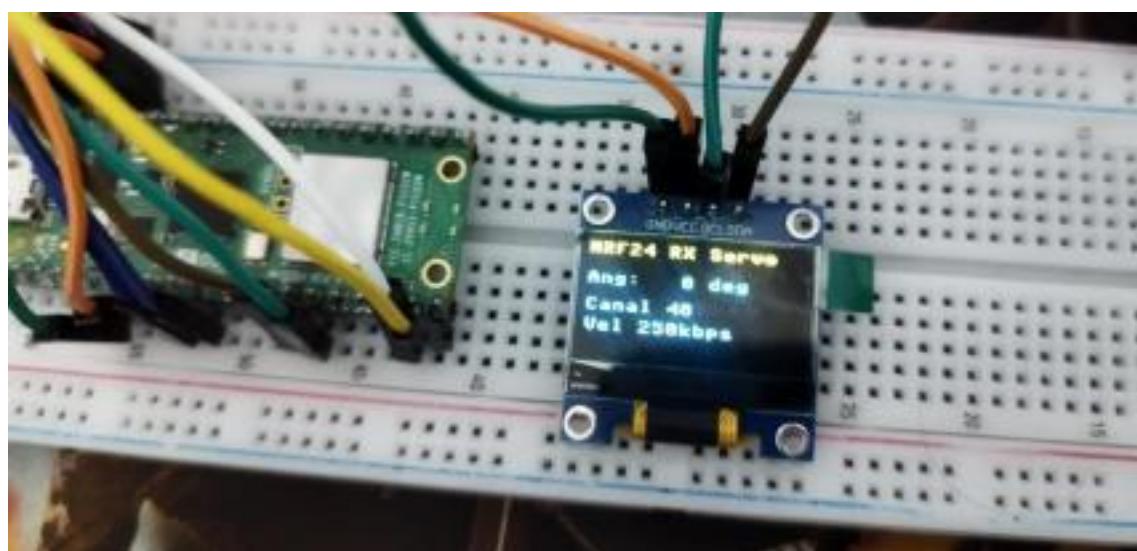
Consola

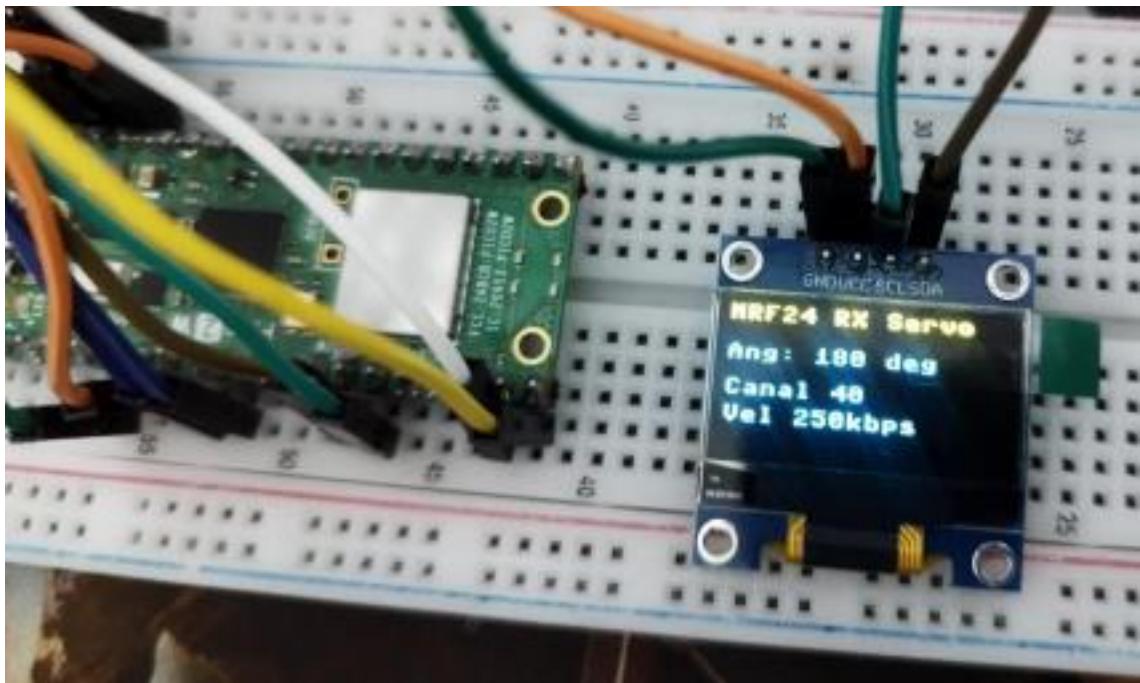
```

RX ang: 140
RX ang: 140
RX ang: 180
RX ang: 180
RX ang: 180
RX ang: 180
RX ang: 0
RX ang: 0
RX ang: 0
RX ang: 137

```

En la oled se refleja cada cambio de Angulo al mover el joystick





#Tx mejorado

```
from machine import Pin, SPI, ADC
import utime, struct
from nrf24l01 import NRF24L01

# ---- Radio SPI0 ----
spi = SPI(0, sck=Pin(6), mosi=Pin(7), miso=Pin(4))
csn = Pin(15, Pin.OUT, value=1)
ce = Pin(14, Pin.OUT, value=0)
nrf.stop_listening()

# ---- Joystick con mapeo directo 0-180° ----
adc_x = ADC(Pin(26))

def leer_angulo_joystick():
    raw = adc_x.read_u16() # 0-65535
    # Mapeo directo: 0=0°, 32767=90°, 65535=180°
    angulo = int((raw * 180) / 65535)
    return max(0, min(180, angulo))

def calcular_checksum(sync, angulo):
    return (sync + (angulo & 0xFF) + ((angulo >> 8) & 0xFF)) & 0xFF
SYNC_BYTE = 0xA5
```

```

angulo_anterior = -1
umbral = 0 # Enviar TODOS los cambios (máxima
respuesta) print("TX Joystick - Rango 0-180° -
2Mbps - Listo")
while True:
    angulo_actual = leer_angulo_joystick()

    # Enviar SIEMPRE (umbral = 0 para máxima respuesta)
    checksum = calcular_checksum(SYNC_BYT
E, angulo_actual)
    paquete = struct.pack("<BHB", SYNC_BYT
E, angulo_actual,
checksum) except Exception as e:
    print(f"X Error: {e}")
    nrf.reg_write(0x07, 0x70)

    utime.sleep_ms(10) # ⚡ SOLO 10ms = 100 FPS!

```

#Rx mejorado

```

from machine import Pin, SPI, I2C, PWM
import utime, struct
from nrf24l01 import NRF24L01
from ssd1306 import SSD1306_I2C

# ---- OLED ----
i2c = I2C(1, scl=Pin(11), sda=Pin(10), freq=400000)
oled = SSD1306_I2C(128, 64, i2c, addr=0x3C)
oled.fill(0)
oled.text("Servo 0-180°", 0, 0)
oled.text("Esperando...", 0, 16)
oled.show()

# ---- Servo con rango completo ----
servo = PWM(Pin(15))
servo.freq(50)

```

```

nrf = NRF24L01(spi, csn, ce, payload_size=4)

TX_ADDR = b'\xE1\xF0\xF0\xF0\xF0'
RX_ADDR = b'\xD2\xF0\xF0\xF0\xF0'

nrf.open_tx_pipe(TX_ADDR)

```

```

nrf.open_rx_pipe(1, RX_ADDR)

# Configuración IDÉNTICA a TX (2Mbps)
nrf.set_power_speed(0, 2) # 0dBm, 2Mbps
nrf.reg_write(0x01, 0x00)
nrf.reg_write(0x04, 0x00)
nrf.reg_write(0x05, 100) # Canal 100

mover_servo_instantaneo(ultimo_angulo)

def verificar_checksum(sync, angulo, checksum):
    calc = (sync + (angulo & 0xFF) + ((angulo >> 8) &
    0xFF)) & 0xFF return checksum == calc

print("□□ RX Servo - 0-180° - 2Mbps - Listo")

# ---- Bucle principal ULTRA RÁPIDO ----
while True:
    if nrf.any():
        try:
            datos = nrf.recv()
            sync, angulo, checksum = struct.unpack("<BHB",
            datos)
            (angulo)
            contador_paquetes += 1

            # Actualizar display solo cada 15 paquetes (no
            bloquear) if contador_paquetes >= 15:
            oled.fill(0)
            oled.text("Servo 0-180°", 0, 0)
            oled.text(f"Angulo: {angulo:3d}°", 0, 16)
            oled.text("Vel: 2Mbps", 0, 32)
            oled.text(f"Pkts: {contador_paquetes}", 0, 48)
            oled.show()
            print(f"□□RX: {angulo}°")
            contador_paquetes = 0

        except Exception as e:
            print(f"Error: {e}")

        # ⚡ Delay mínimo para máxima velocidad

```

```

utime.sleep_us(500) # 0.5ms

from machine import Pin, SPI, I2C, PWM
import utime, struct
from nrf24l01 import NRF24L01
from ssd1306 import SSD1306_I2C

oled.text("Servo 0-180°", 0, 0)
oled.text("Esperando...", 0, 16)
oled.show()
# ---- Servo con rango completo ----
servo = PWM(Pin(15))
servo.freq(50)

def mover_servo_instantaneo(angulo):
    """Mueve el servo INSTANTÁNEAMENTE a la
    posición"""" angulo = max(0, min(180,
    int(angulo)))
    # Conversión directa a pulso PWM
    pulso_us = 500 + (angulo * 2000) // 180
    servo.duty_ns(pulso_us * 1000)

# ---- Radio MÁXIMA VELOCIDAD ----
spi = SPI(0, sck=Pin(6), mosi=Pin(7), miso=Pin(4))
csn = Pin(5, Pin.OUT, value=1)
ce = Pin(14, Pin.OUT, value=0)
nrf.open_rx_pipe(1, RX_ADDR)

# Configuración IDÉNTICA a TX (2Mbps)
nrf.set_power_speed(0, 2) # 0dBm, 2Mbps
nrf.reg_write(0x01, 0x00)
nrf.reg_write(0x04, 0x00)
nrf.reg_write(0x05, 100) # Canal 100
nrf.start_listening()

# ---- Variables ----
SYNC_BYTE = 0xA5
ultimo_angulo = 90
ultima_actualizacion = 0
contador_paquetes = 0

```

```

mover_servo_instantaneo(ultimo_angulo)

def verificar_checksum(sync, angulo, checksum):
    calc = (sync + (angulo & 0xFF) + ((angulo >> 8) &
    0xFF)) & 0xFF return checksum == calc

print("□□ RX Servo - 0-180° - 2Mbps - Listo")

# ---- Bucle principal ULTRA RÁPIDO ----
while True:
):

# ↳ MOVER SERVO
INMEDIATAMENTE
mover_servo_instantaneo(angulo)
contador_paquetes += 1

# Actualizar display solo cada 15 paquetes (no
bloquear) if contador_paquetes >= 15:
", 0, 32)
oled.text(f"Pkts: {contador_paquetes}", 0, 48)
oled.show()
print(f"□□RX: {angulo}°")
contador_paquetes = 0

except Exception as e:
print(f"Error: {e}")

# ↳ Delay mínimo para máxima velocidad
utime.sleep_us(500) # 0.5ms

```

El servo se mueve a cada posición, pero **no se queda quieto** porque es un **servo de rotación continua**.

En un servo de rotación continua:

- **1500us = STOP** (no se mueve)
- **<1500us = Gira en una dirección**
- **>1500us = Gira en la otra dirección**

TX con OLED- muestra datos ANGULOS

```
#Tx mejorado - EXACTO AL ORIGINAL + OLED
from machine import Pin, SPI, ADC, I2C
import utime, struct
from nrf24l01 import NRF24L01
from ssd1306 import SSD1306_I2C

# ---- OLED (I2C1: GP11 SCL, GP10 SDA) ----
i2c = I2C(1, scl=Pin(11), sda=Pin(10), freq=400000)
oled = SSD1306_I2C(128, 64, i2c, addr=0x3C)
RX_ADDR = b'\xD2\xF0\xF0\xF0\xF0'

nrf.open_tx_pipe(TX_ADDR)
nrf.open_rx_pipe(1, RX_ADDR)

# Configuración MÁXIMA velocidad
nrf.set_power_speed(0, 2) # 0dBm, 2Mbps (MÁXIMA
VELOCIDAD) nrf.reg_write(0x01, 0x00) # No Auto-Ack
nrf.reg_write(0x04, 0x00) # No retransmisiones
nrf.reg_write(0x05, 100) # Canal 100 (menos interferencia)
nrf.stop_listening()

# ---- Joystick con mapeo directo 0-180° ----
adc_x = ADC(Pin(26))
umbral = 0 # Enviar TODOS los cambios (máxima respuesta)

# Variable SOLO para OLED - NO INTERFIERE
ultimo_angulo_oled = 0

print("TX Joystick - Rango 0-180° - 2Mbps - Listo")

while True:
    angulo_actual = leer_angulo_joystick()

    # Enviar SIEMPRE (umbral = 0 para máxima respuesta) - CÓDIGO ORIGINAL
    EXACTO checksum = calcular_checksum(SYNC_BYT
E, angulo_actual)
    paquete = struct.pack("<BHB", SYNC_BYT
E, angulo_actual,
checksum)
    try:
        nrf.send(paquete)
        if angulo_actual != angulo_anterior:
```

```

COMPLETAMENTE INDEPENDIENTE
# Se ejecuta en paralelo, NO afecta el timing original
if angulo_actual != ultimo_angulo_oled:
    oled.fill(0)
    oled.text("CONTROL TX", 0, 0)
    oled.text(f"Angulo: {angulo_actual:3d}°", 0, 16)
    oled.text("2Mbps", 0, 32)
    oled.show()
    ultimo_angulo_oled = angulo_actual

```

utime.sleep_ms(10) # ↪ SOLO 10ms = 100 FPS! - EXACTO AL ORIGINAL

Txt muestra mensaje en oled

```

#Tx mejorado - ORIGINAL EXACTO + OLED FIJA
from machine import Pin, SPI, ADC, I2C
import utime, struct
from nrf24l01 import NRF24L01
from ssd1306 import SSD1306_I2C
", 0, 48)
oled.show()

```

```

# ---- Radio SPI0 ----
spi = SPI(0, sck=Pin(6), mosi=Pin(7), miso=Pin(4))
csn = Pin(15, Pin.OUT, value=1)
ce = Pin(14, Pin.OUT, value=0)

```

```

nrf = NRF24L01(spi, csn, ce, payload_size=4)
TX_ADDR = b'\xE1\xF0\xF0\xF0\xF0'
RX_ADDR = b'\xD2\xF0\xF0\xF0\xF0'

```

```

nrf.open_tx_pipe(TX_ADDR)
nrf.

```

```

# ---- Joystick con mapeo directo 0-180° ----
adc_x = ADC(Pin(26))

```

```

def leer_angulo_joystick():
    raw = adc_x.read_u16() # 0-65535

```

```

# Mapeo directo: 0=0°, 32767=90°, 65535=180°
angulo = int((raw * 180) / 65535)
return max(0, min(180, angulo))

def calcular_checksum(sync, angulo):
    return (sync + (angulo & 0xFF) + ((angulo >> 8) & 0xFF)) & 0xFF

SYNC_BYTEx = 0xA5
angulo_anterior = -1
umbral = 0 # Enviar TODOS los cambios (máxima respuesta)

print("TX Joystick - Rango 0-180° - 2Mbps - Listo")

while True:
    angulo_actual = leer_angulo_joystick()

    = angulo_actual
    except Exception as e:
        print(f"X Error: {e}")
        nrf.reg_write(0x07, 0x70)

    utime.sleep_ms(10) # ⚡ SOLO 10ms = 100 FPS!

```

RX con acelerómetro- muestra datos en consola

```

# ===== RX con acelerometro completo (ax, ay, az, |a|)
===== from machine import Pin, SPI, I2C, PWM
import utime, struct
from nrf24l01 import NRF24L01
oled = SSD1306_I2C(128, 64, i2c1, addr=0x3C)
oled.fill(0)
oled.text("Servo 0-180°", 0, 0)
oled.text("Esperando...", 0, 16)
oled.show()
# ---- Servo ----
servo = PWM(Pin(15))
servo.freq(50)
def mover_servo_instantaneo(angulo):

```

```

angulo = max(0, min(180, int(angulo)))
pulso_us = 500 + (angulo * 2000) // 180
servo.duty_ns(pulso_us * 1000)

payload_size=4)

TX_ADDR = b'\xE1\xF0\xF0\xF0\xF0'
RX_ADDR = b'\xD2\xF0\xF0\xF0\xF0'

nrf.open_tx_pipe(TX_ADDR)
nrf.open_rx_pipe(1, RX_ADDR)
nrf.set_power_speed(0, 2)
nrf.reg_write(0x01, 0x00)
nrf.reg_write(0x04, 0x00)
nrf.reg_write(0x05, 100)
nrf.start_listening()

# ---- Acelerómetro MPU6050 (I2C0: GP17 SCL, GP16 SDA, AD0=GND →
# 0x68) ---- i2c0 =
REG_AX_H = 0x3B
LSB_PER_G = 16384.0
G = 9.80665

def mpu_write(reg, val):
    i2c0.writeto_mem(MPU_ADDR, reg, bytes([val]))

def mpu_read(reg, n):
    return i2c0.readfrom_mem(MPU_ADDR, reg, n)

def mpu_init():
    mpu_write(REG_PWR1, 0x00)
    utime.sleep_ms(100)

def mpu_read_accel_mps2():
    d = mpu_read(REG_AX_H, 6)
    ax = (d[0]<<8) | d[1]
    ay = (d[2]<<8) | d[3]
    az = (d[4]<<8) | d[5]
    if ax & 0x8000: ax -= 65536
    if ay & 0x8000: ay -= 65536
    if az & 0x8000: az -= 65536
    ax_mps2 = (ax / LSB_PER_G) * G

```

```

ay_mps2 = (ay / LSB_PER_G) * G
az_mps2 = (az / LSB_PER_G) * G
return ax_mps2, ay_mps2, az_mps2

# ---- Calibración de offsets (2s en reposo) ----
N = 0
t0 = utime.ticks_ms()
while utime.ticks_diff(utime.ticks_ms(), t0) < 2000:
    ax, ay, az = mpu_read_accel_mps2()
    sx += ax; sy += ay; sz += az
    N += 1
    utime.sleep_ms(10)
off_ax = sx / N
off_ay = sy / N
off_az = (sz / N) - G # restar gravedad en Z
print("Offsets:", off_ax, off_ay, off_az)

# ---- Variables de RX ----
SYNC_BYTE = 0xA5
contador_paquetes = 0
mover_servo_instantaneo(90)
print("RX Servo + MPU6050 listo (2Mbps)")

def verificar_checksum(sync, angulo, checksum):
    calc = (sync + (angulo & 0xFF) + ((angulo >> 8) &
    0xFF)) & 0xFF return checksum == calc

# ---- Bucle principal ----
and verificar_checksum(sync, angulo, checksum) and 0 <=
angulo <= 180: mover_servo_instantaneo(angulo)
contador_paquetes += 1

# Leer acelerómetro
ax, ay, az = mpu_read_accel_mps2()
ax -= off_ax
ay -= off_ay
az -= off_az
a_total = math.sqrt(ax**2 + ay**2 + az**2)

# Actualizar OLED cada 15 paquetes
if contador_paquetes >= 15:
    oled.fill(0)

```

```
oled.text("Servo 0-180°", 0, 0)
oled.text(f"Angulo: {angulo:3d}°", 0, 16)
oled.text("Vel: 2Mbps", 0, 32)
oled.text(f"Pkts: {contador_paquetes}", 0, 48)
oled.show()

# Consola con 4 columnas (ángulo y 3 ejes + módulo)
print(f"ANG:{angulo:3d}° | ax:{ax:7.3f} | ay:{ay:7.3f} | az:{az:7.3f} |
|a|:{a_total:7.3f}")
contador_paquetes = 0

except Exception as e:
    print("Error:", e)

utime.sleep_us(500)
```

```
123     mover_servo_instantaneo(angulo)
124     contador_paquetes += 1
125
126     # Leer acelerómetro
127     ax, ay, az = mpu_read_accel_mps2()
128     ax -= off_ax
129     ay -= off_ay
130     az -= off_az
131     a_total = math.sqrt(ax**2 + ay**2 + az**2)
132
```

Consola

```
ANG:140° | ax: 0.030 | ay: 0.044 | az: 9.832 | |al: 9.832
ANG:140° | ax: -0.030 | ay: 0.003 | az: 9.794 | |al: 9.794
ANG:140° | ax: -0.034 | ay: -0.088 | az: 9.810 | |al: 9.811
ANG:140° | ax: 0.076 | ay: -0.026 | az: 9.755 | |al: 9.755
ANG:180° | ax: 0.067 | ay: -0.016 | az: 9.827 | |al: 9.827
ANG:140° | ax: -0.089 | ay: -0.021 | az: 9.841 | |al: 9.841
ANG:141° | ax: -0.026 | ay: 0.036 | az: 9.777 | |al: 9.777
ANG:140° | ax: 0.023 | ay: 0.084 | az: 9.803 | |al: 9.803
ANG:141° | ax: 0.016 | ay: 0.027 | az: 9.734 | |al: 9.734
```

