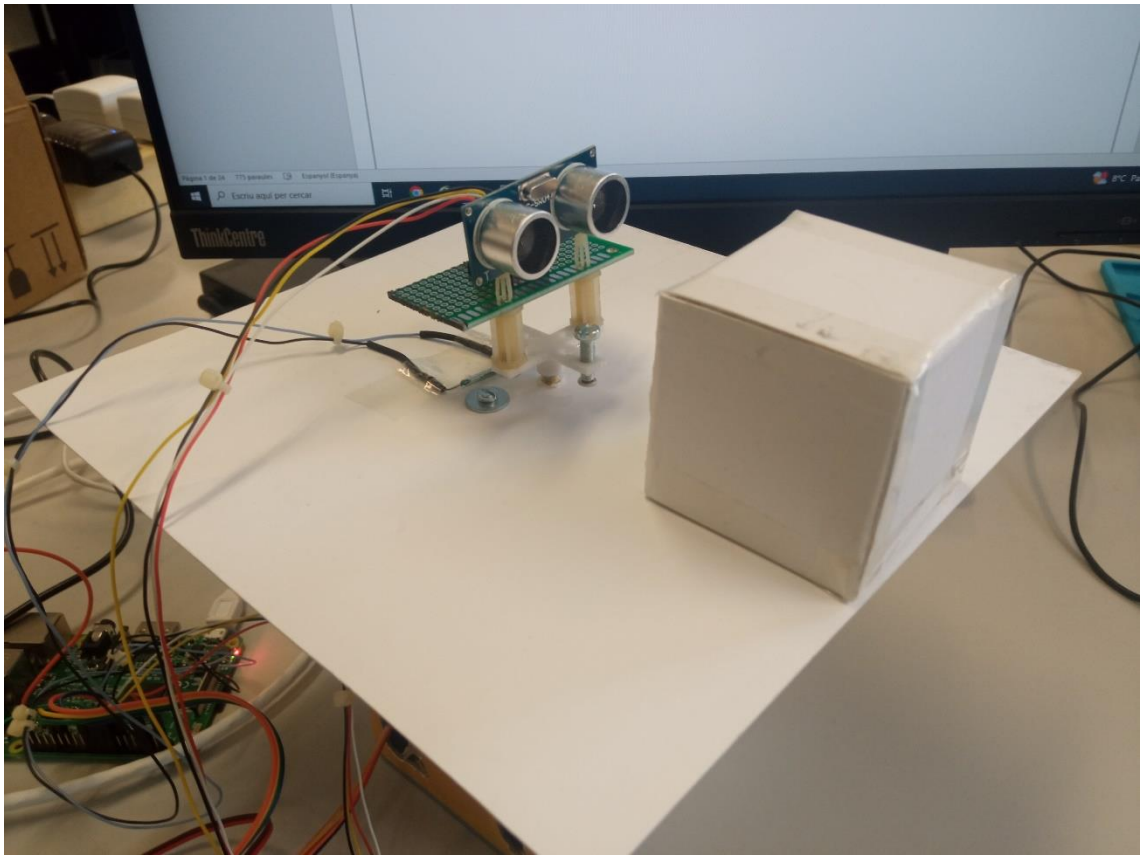
BUSCAMINAS



Albert Llopis

INDICE

1.DESCRIPCIÓN:

Objetivo del Juego:

Este juego está inspirado en el mítico juego del buscaminas, pero extrapolándolo a la realidad. Una versión en realidad 3D.

El programa escoge al azar una casilla, donde depositará la "BOMBA" y el jugador deberá mediante el "CUBO", averiguar en qué casilla se haya. Una vez localizada, esta explotara.

Dando así por terminado el juego. El reto consiste en detectar la mina en los menos movimientos posibles. Lograrlo en un solo movimiento es el objetivo máximo del juego.

Objetivo del Técnico:

Realizar el proyecto en 3 versiones distintas de programación, para explorar las capacidades de Raspberry PI 3 B++ .

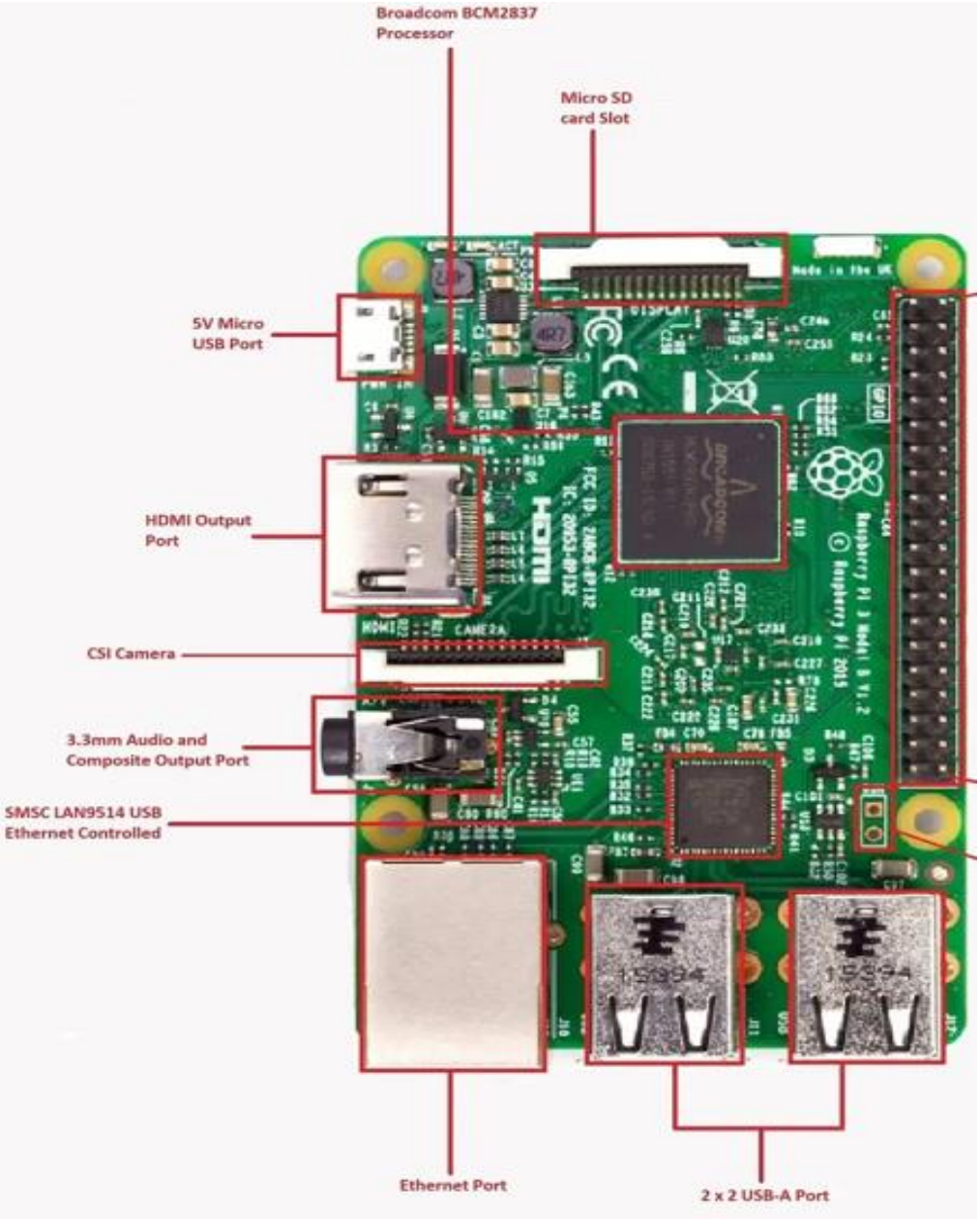
Modos: Secuenciador, MultiHilo, MultiProceso.

2. DESCRIPCIÓN TÉCNICA:

2.1. Hardware:

- 2.1.1 Paspberry Pi B+
- 2.1.2 FA: HW131
- 2.1.3 Motor (Paso a Paso): 28BY-48 [5V DC]
- 2.1.4 Driver Motor: VLN 2003
- 2.1.5 Ultrasonido: HC-SR04
- 2.1.6 Pulsador Final Carrera:
- 2.1.7 PC

2.1.1. Paspberry Pi B+:



Descripción:

Una Raspberry Pi es un pequeño ordenador que tiene un tamaño similar a una placa de Arduino y que también se utiliza a menudo en el mundo maker y la enseñanza para construir proyectos de electrónica.

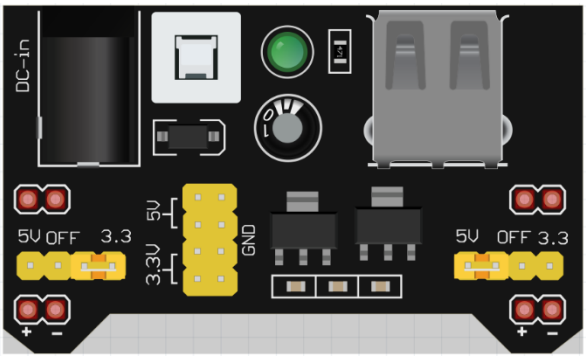
La Raspberry Pi 3 Modelo B+ es una versión mejorada de la Raspberry Pi 3 Modelo B. Se basa en el sistema en chip (SoC) BCM2837B0, que incluye un procesador ARMv8 de 64 bits de cuatro núcleos a 1,4 GHz y una potente GPU VideoCore IV.

Su sistema operativo Debian, basado en Linux. Y nuestros programas serán programados en Python.

Configuración PIN's:

Las múltiples opciones que ofrecen los pin's de trabajo le otorgan una flexibilidad, tanto en frecuencias distintas, funciones como en tipo de comportamientos (in, out, Vcc, Gnd…).

2.1.2. FA HW131

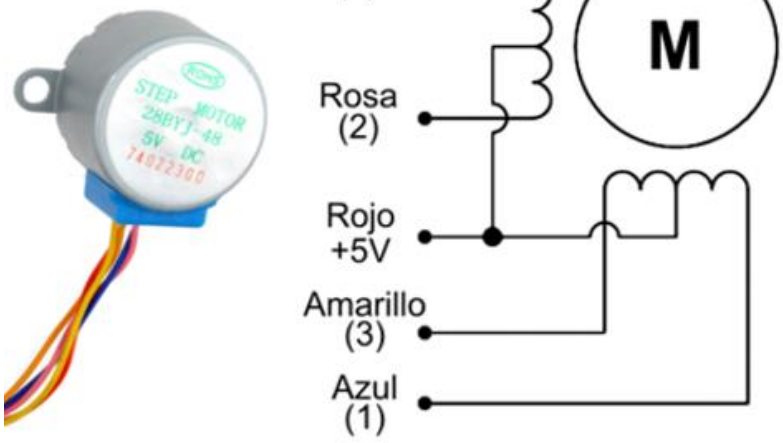


- *Descripción*:

Sera nuestra FA para alimentar el funcionamiento de nuestro Motor Paso a Paso.

Fuente para protoboard con salidas de voltaje seleccionable de 3.3V y/o 5V, son reguladores AMS1117 de 3.3 y 5.0 voltios capaces de otorgar un maximo de 1 Amperio en su salida. Alimentacion via Jack-DC de 7 a 12 voltios.

2.1.3. Motor (Paso a Paso): 28BY-48 [5V DC]



- *Descripción:*

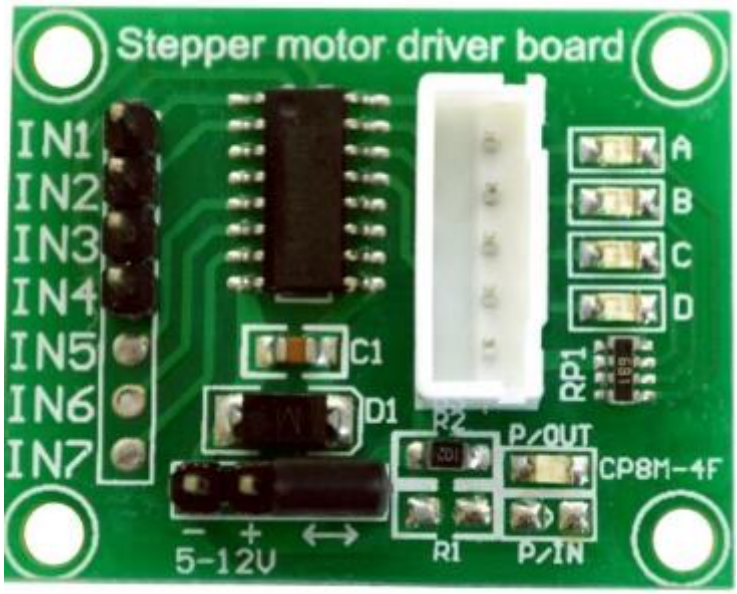
Con este motor moveremos (sobre su propio eje Y), el medidor de distancias.

Es un motor unipolar con las siguientes características:

- Tensión nominal de entre 5V y 12 V.

- 4 Fases.
- Resistencia 50 Ω.
- Par motor de 34 Newton / metro más o menos 0,34 Kg por cm.
- Consumo de unos 55 mA.
- 8 pasos por vuelta.
- Reductora de 1 / 64.

2.1.4. Driver Motor: VLN 2003



- Descripción:

Con este driver controlaremos las bobinas del motor, tanto sentido de giro, como velocidad (rpm). A su vez este será controlado por nuestro programa.

- Configuración pins:

- IN1 …IN4: Orden de activado de las bobinas del Motor.

- Podemos activarlas de tres modalidades: Paso, Paso Completo, Medio Paso.

2.1.5. Ultrasonido: HC-SR04

- *Descripción:*

  Sensor de distancias entre 2 y 450 cm. Sensor por ultrasonido, este irá montado en el eje del Motor. A su vez nos localizara el "CUBO", objeto con el que intentaremos localizar la "BOMBA".

- *Config pins:*

  Vcc: +5V

  Gnd: 0V

  Trig: Activamos medición.

  Echo: Recibimos respuesta si localiza objeto.

2.1.6. Detecctor "Pos_Cero":

- Descripción:

Interruptor NA, con activación magnética. Con este interruptor indicaremos al programa donde se haya el cero mecánico. Para sincronizarlo con las posiciones de las casillas.

2.1.7. PC

- Descripción:

Ordenador personal destinado a comunicarnos con las Raspberry y poder cargar nuestros programas, una vez editados.

2.2. SOFTWARE:

2.2.1. SECUENCIADOR

2.2.2. MULTIHILOS

2.2.3. MULTIPROCESSING

Este proyecto consta de 17 evoluciones (Mains, Clases), pero solo presentaremos la última versión de cada tipo de programación (Secuenciador, Multihilo, Multiprocesssing).

2.2.1 SECUENCIADOR:

Programa principal (Main) que hará la llamada a las librerías necesarias para trabajar, así como a las clases con la que trabaja y que hemos creado especialmente para este proyecto.

Secuenciador: "BUSCAMINAS_MAIN_V4"

```
import RPi.GPIO as GPIO
import time
import os

from Motor_pp_CLASE_v3 import *          #


#-------------------------------------------------------[DECLARACIÓN VARIABLES]
Final_Ciclo = False


#-------------------------------------------------------[PIN's RASPBERRY]
IN1 = 4      # GPIO 4
IN2 = 5      # GPIO 5
IN3 = 6      # GPIO 6
IN4 = 27     # GPIO 27

#-------------------------------------------------------[OBJETOS]
Motor_1 = Motor(IN1, IN2, IN3, IN4)      # Motor_1




#-------------------------------------------------------[MAIN]
try:
    while Final_Ciclo == False:
        Final_Ciclo = Motor_1.Giro_R()


#-------------------------------------------------------[INTERRUPCIONES]
except KeyboardInterrupt:

   Motor_1.cleanup()
```

Secuenciador: *"BUSCAMINAS_CLASE_V4"*

```python
import RPi.GPIO as GPIO
import time

class Motor:
    def __init__(self, IN1, IN2, IN3, IN4):     # CONSTRUCTOR

        self.IN1 = IN1
        self.IN2 = IN2
        self.IN3 = IN3
        self.IN4 = IN4

        GPIO.setmode(GPIO.BCM)                  # Configurar els pins GPIO

        GPIO.setup  (self.IN1,   GPIO.OUT)      # IN1: SALIDA
        GPIO.setup  (self.IN2,   GPIO.OUT)      # IN2: SALIDA
        GPIO.setup  (self.IN3,   GPIO.OUT)      # IN3: SALIDA
        GPIO.setup  (self.IN4,   GPIO.OUT)      # IN4: SALIDA


#-----------------------------------------------[GIROS MOTOR]
    def Giro_R(self):                           # "giro_derecha"



        for rpm in range (512):

            GPIO.output(self.IN1,GPIO.HIGH)     # IN1 --> "1"
            GPIO.output(self.IN2,GPIO.LOW)      # IN2 --> "0"
            GPIO.output(self.IN3,GPIO.LOW)      # IN3 --> "0"
            GPIO.output(self.IN4,GPIO.LOW)      # IN4 --> "0"
            time.sleep(0.020)                   # Retardo (20 mSeg)

            GPIO.output(self.IN1,GPIO.LOW)      # IN1 --> "0"
            GPIO.output(self.IN2,GPIO.HIGH)     # IN2 --> "1"
            GPIO.output(self.IN3,GPIO.LOW)      # IN3 --> "0"
            GPIO.output(self.IN4,GPIO.LOW)      # IN4 --> "0"
            time.sleep(0.020)                   # Retardo (20 mSeg)

            GPIO.output(self.IN1,GPIO.LOW)      # IN1 --> "0"
            GPIO.output(self.IN2,GPIO.LOW)      # IN2 --> "0"
            GPIO.output(self.IN3,GPIO.HIGH)     # IN3 --> "1"
            GPIO.output(self.IN4,GPIO.LOW)      # IN4 --> "0"
            time.sleep(0.020)                   # Retardo (20 mSeg)

            GPIO.output(self.IN1,GPIO.LOW)      # IN1 --> "0"
            GPIO.output(self.IN2,GPIO.LOW)      # IN2 --> "0"
            GPIO.output(self.IN3,GPIO.LOW)      # IN3 --> "0"
            GPIO.output(self.IN4,GPIO.HIGH)     # IN4 --> "1"
            time.sleep(0.020)                   # Retardo (20 mSeg)

        return True

    def Giro_R_Matriz(self):                    # "giro_derecha"
```

```
    def Giro_R_Matriz(self):                    # "giro_derecha"


        for rpm in range (512):

            GPIO.output(self.IN1,GPIO.HIGH)     # IN1 --> "1"
            GPIO.output(self.IN2,GPIO.LOW)      # IN2 --> "0"
            GPIO.output(self.IN3,GPIO.LOW)      # IN3 --> "0"
            GPIO.output(self.IN4,GPIO.LOW)      # IN4 --> "0"
            time.sleep(0.020)                   # Retardo (20 mSeg)

            GPIO.output(self.IN1,GPIO.LOW)      # IN1 --> "0"
            GPIO.output(self.IN2,GPIO.HIGH)     # IN2 --> "1"
            GPIO.output(self.IN3,GPIO.LOW)      # IN3 --> "0"
            GPIO.output(self.IN4,GPIO.LOW)      # IN4 --> "0"
            time.sleep(0.020)                   # Retardo (20 mSeg)

            GPIO.output(self.IN1,GPIO.LOW)      # IN1 --> "0"
            GPIO.output(self.IN2,GPIO.LOW)      # IN2 --> "0"
            GPIO.output(self.IN3,GPIO.HIGH)     # IN3 --> "1"
            GPIO.output(self.IN4,GPIO.LOW)      # IN4 --> "0"
            time.sleep(0.020)                   # Retardo (20 mSeg)

            GPIO.output(self.IN1,GPIO.LOW)      # IN1 --> "0"
            GPIO.output(self.IN2,GPIO.LOW)      # IN2 --> "0"
            GPIO.output(self.IN3,GPIO.LOW)      # IN3 --> "0"
            GPIO.output(self.IN4,GPIO.HIGH)     # IN4 --> "1"
            time.sleep(0.020)                   # Retardo (20 mSeg)

        return True

#-----------------------------------------[SALIDA PROGRAMA]
    def cleanup(self):                          # Limpieza Conf. Pins
        GPIO.output(self.IN1,GPIO.LOW)          # IN1 --> "0"
        GPIO.output(self.IN2,GPIO.LOW)          # IN2 --> "0"
        GPIO.output(self.IN3,GPIO.LOW)          # IN3 --> "0"
        GPIO.output(self.IN4,GPIO.LOW)          # IN4 --> "0"

        time.sleep(1)

        GPIO.cleanup()
```

2.2.2. MULTIHILOS

MAIN: "BUSCAMINAS_MAIN_V17"

```
import pigpio   # Control GPIO
import time     # Control Tiempo
import os       # Obciones Sistema Operativo

from threading            import Thread      #  MULTIHILO

from Motor_pp_CLASE_v13   import *           #  MOTOR
from Distancia_CLASE_v13  import *           #  DISTANCIA
from Juego_CLASE_v17      import *           #  JUEGO


#-------------------------------------------------------------------------------------------------------[DECLARACIÓN VARIABLES]
Final_Ciclo = False


#-------------------------------------------------------------------------------------------------------[PIN's RASPBERRY]
IN  = (4, 5, 6, 27)                                 # MOTOR_1

                                                    # DISTANCIA _X
TRIG_PIN = 23                                       # GPIO23 Activa Detector   (Trigger)
ECHO_PIN = 24                                       # GPIO24 Objeto Detectado  (Echo)

Pos_Cero = 25                                       # GPIO25 Motor en 0, mecánico.

Pul_Start = 22                                      # GPIO22 Pulsador Start jugador.

#-------------------------------------------------------------------------------------------------------[OBJETOS]
pi = pigpio.pi()                                    # Conexión al "daemon pigpio"

Motor_1 = Motor(pi, IN)                             # Motor_1

Distancia_1 = SensorDistancia (pi, TRIG_PIN, ECHO_PIN)  # Distancia_1

Juego_1 = Juego(pi, Pos_Cero, Pul_Start)            # Juego_1

#-------------------------------------------------------------------------------------------------------[FUNCIONES]
def Mov_Motor (Motor_1):                                # MOTOR
    global Final_Ciclo                                  # Variable global, para salir del While (MAIN)



def Med_Distancia (Distancia_1):                        # DISTANCIA
    global dist_x

    while Final_Ciclo == False:
        dist_x = Distancia_1.mesura_distancia()         # Hago mediciones


        if dist_x > 0:

            time.sleep(1)                               # Tiempo reposo (1 seg)



def Ini_Juego (Juego_1):                                    # INICIO JUEGO

    os.system('clear')                                      # Limpia pantalla
    PosBomba = Juego_1.Pos_Bomba()                          # Donde estará la BOMBA

    while Juego_1.Pul_Start() == True:                      # Mientras selector Start = True

        while Juego_1.Pos_Cero() == False:                  # Mientras no llege a Pos_Cero

            os.system('clear')                              # Limpia pantalla
            print ("\n....Buscando POS REF....")            # Mensaje

            Motor_1.Giro_L_Paso_C (1)                       # Gira a la izquierda

        print(f"La BOMBA esta en la Posición: {PosBomba}")# Mensaje

        print ("\nPOSICIÓN 1")                             # POSICIÓN 1 (0°)
        time.sleep(1)                                       # Tiempo-antivibraciones
        Juego_1.Check_Cajita(dist_x,1,PosBomba)             # Busca si hay CUBO, esta en Pos 1,
                                                            # envia Pos_Bomba

        print ("\nPOSICIÓN 2")                             # POSICIÓN 2 (45°)
        Motor_1.Giro_R_Paso_C (45)                          # Mismo proceso que Pos 1
        time.sleep(1)
        Juego_1.Check_Cajita(dist_x,2,PosBomba)

        print ("\nPOSICIÓN 3")                             # POSICIÓN 3 (90°)
        Motor_1.Giro_R_Paso_C (45)                          # Mismo proceso que Pos 1
        time.sleep(1)
        Juego_1.Check_Cajita(dist_x,3,PosBomba)
```

```
#-------------------------------------------------------------------------------------------------------[MAIN]
if __name__ == '__main__':

    try:
        while Final_Ciclo == False:                                             # Final de Ciclo

            lista_threads = [
                            Thread(target = Mov_Motor,      args = (Motor_1,)),     # Creo un Hilo ---> "lista_threads[0]" --> Motor_1
                            Thread(target = Med_Distancia,  args = (Distancia_1,)), # Creo un Hilo ---> "lista_threads[1]" --> Distancia_1
                            Thread(target = Ini_Juego,      args = (Juego_1,))      # Creo un Hilo ---> "lista_threads[0]" --> Juego_1
                            ]


            lista_threads[0].start()                                            # Inicio preceso Hilo [0] --> Motor_1
            lista_threads[1].start()                                            # Inicio preceso Hilo [1] --> Distancia_1
            lista_threads[2].start()                                            # Inicio preceso Hilo [1] --> Juego_1

            lista_threads[0].join ()                                            # Espero Hilo [0] --> Motor_1
            lista_threads[1].join ()                                            # Espero Hilo [1] --> Distancia_1
            lista_threads[2].join ()                                            # Espero Hilo [2] --> Juego_1


#-------------------------------------------------------------------------------------------------------[INTERRUPCIONES]
except KeyboardInterrupt:


    print ("\n.....Parando Hilos.......")                                       # Mensaje

    pi.stop()                                                                   # Desconecta control pin's

    print ("\n.....Hilos Parados.......")                                      # Mensaje
```

CLASE Distancia:"Distancia_CLASE_v13"

```
import pigpio   # Control GPIO
import time

class SensorDistancia:                                          # CONSTRUCTOR()
    def __init__(self, pi, trigger_pin, echo_pin):

        self.pi = pi                                            # instancia de pigpio

        self.trigger_pin = trigger_pin                          # traspaso valor de variables
        self.echo_pin    = echo_pin                             # traspaso valor de variables

        self.pi.set_mode (trigger_pin, pigpio.OUTPUT)           # trigger_pin: SALIDA
        self.pi.set_mode (echo_pin,    pigpio.INPUT )           # echo_pin    : ENTRADA

    def mesura_distancia(self):
        """Mesura la distància en centímetres."""
        self.pi.write(self.trigger_pin, 1)              # Envia un pols de 10 µs al pin trigger
        time.sleep(0.00001)
        self.pi.write(self.trigger_pin, 0)

        start_time = None
        stop_time = None

        timeout_start = time.time()                     # ESPERA QUE COMIENCE ECHO
        while self.pi.read(self.echo_pin) == 0:
            start_time = time.time()
                                                        # Evita bloqueig si no arriba el senyal
            if start_time - timeout_start > 0.1:        # Timeout de 100 ms
                return -1                               # Error: sense senyal

        timeout_start = time.time()                     # ESPERA QUE ACABE ECHO
        while self.pi.read(self.echo_pin) == 1:
            stop_time = time.time()

                                                        # Evita bloqueig si l'eco és massa llarg
            if stop_time - timeout_start > 0.1:         # Timeout de 100 ms
                return -2                               # Error: eco massa llarg

        if start_time is None or stop_time is None:     # Comprova que start_time i stop_time s'han definit
            return -3                                   # Error: mesura no vàlida

        elapsed_time = stop_time - start_time           # Calcula el temps i la distància
        distancia = (elapsed_time * 34300) / 2          # cm

        return distancia
```

CLASE Motor:"Motor_CLASE_v13"

```
import pigpio   # Control GPIO
import time

class Motor:
    def __init__(self, pi, input):                                          # CONSTRUCTOR

        self.pi = pi        # instancia de pigpio

        self.input = input  # (IN1, IN2, IN3, IN4: Pines de Control Motor, R_L)

        #.....................................................................# PIN's RASPBERRY
        self.pi.set_mode (input[0], pigpio.OUTPUT)   # IN1: SALIDA
        self.pi.set_mode (input[1], pigpio.OUTPUT)   # IN2: SALIDA
        self.pi.set_mode (input[2], pigpio.OUTPUT)   # IN3: SALIDA
        self.pi.set_mode (input[3], pigpio.OUTPUT)   # IN4: SALIDA

        #.....................................................................# CONFIG. MOTOR (GIROS: R_L)
        self.Paso_R = [                         # Config. Motor --> Paso_R       # [MATRIZ]
                       [1,0,0,0],
                       [0,1,0,0],
                       [0,0,1,0],
                       [0,0,0,1]
                       ]

        self.Paso_L = [                         # Config. Motor --> Paso_L
                       [0,0,0,1],
                       [0,0,1,0],
                       [0,1,0,0],
                       [1,0,0,0]
                       ]


#.....................................................................................
self.Paso_C_R = [                               # Config. Motor --> Paso Completo_R
                 [1,1,0,0],
                 [0,1,1,0],
                 [0,0,1,1],
                 [1,0,0,1]
                 ]

self.Paso_C_L = [                               # Config. Motor --> Paso Completo_L
                 [0,0,1,1],
                 [0,1,1,0],
                 [1,1,0,0],
                 [1,0,0,1]
                 ]


#.....................................................................................
self.Paso_MP_R = [                              # Config. Motor --> Paso Medio Paso R
                  [1,0,0,0],
                  [1,1,0,0],
                  [0,1,0,0],
                  [0,1,1,0],
                  [0,0,1,0],
                  [0,0,1,1],
                  [0,0,0,1],
                  [1,0,0,1]
                 ]

self.Paso_MP_L = [                              # Config. Motor --> Paso Medio Paso L
                  [0,0,0,1],
                  [0,0,1,1],
                  [0,0,1,0],
                  [0,1,1,0],
                  [0,1,0,0],
                  [1,1,0,0],
                  [1,0,0,0],
                  [1,0,0,1]
                 ]
```

```
#.....................................................................................
self.Paso_MP_R = [                          # Config. Motor --> Paso Medio Paso R
                 [1,0,0,0],
                 [1,1,0,0],
                 [0,1,0,0],
                 [0,1,1,0],
                 [0,0,1,0],
                 [0,0,1,1],
                 [0,0,0,1],
                 [1,0,0,1]
                 ]

self.Paso_MP_L = [                          # Config. Motor --> Paso Medio Paso L
                 [0,0,0,1],
                 [0,0,1,1],
                 [0,0,1,0],
                 [0,1,1,0],
                 [0,1,0,0],
                 [1,1,0,0],
                 [1,0,0,0],
                 [1,0,0,1]
                 ]
```

```
#*************************************[FUNCIONES GIROS MOTOR]*************************************************************
                                                                                #HIGH = 1  --> No acepta HIGH
                                                                                #LOW = 0   --> No acepta LOW
    def Giro_R(self, Grados):#.....................................................................# "Giro_Derecha"
                                                                                  # [SECUENCIAL]
        nrpm = self.Conv_Grad_rpm(Grados)                   # Paso de Grados --> rpm

        for rpm in range (nrpm):

            self.pi.write(self.input[0], 1 )                # IN1 --> "1"
            self.pi.write(self.input[1], 0 )                # IN2 --> "0"
            self.pi.write(self.input[2], 0 )                # IN3 --> "0"
            self.pi.write(self.input[3], 0 )                # IN4 --> "0"
            time.sleep(0.010)                                 # Retardo (10 mSeg)

            self.pi.write(self.input[0], 0 )                # IN1 --> "0"
            self.pi.write(self.input[1], 1 )                # IN2 --> "1"
            self.pi.write(self.input[2], 0 )                # IN3 --> "0"
            self.pi.write(self.input[3], 0 )                # IN4 --> "0"
            time.sleep(0.010)                                 # Retardo (10 mSeg)

            self.pi.write(self.input[0], 0 )                # IN1 --> "0"
            self.pi.write(self.input[1], 0 )                # IN2 --> "0"
            self.pi.write(self.input[2], 1 )                # IN3 --> "1"
            self.pi.write(self.input[3], 0 )                # IN4 --> "0"
            time.sleep(0.010)                                 # Retardo (10 mSeg)

            self.pi.write(self.input[0], 0 )                # IN1 --> "0"
            self.pi.write(self.input[1], 0 )                # IN2 --> "0"
            self.pi.write(self.input[2], 0 )                # IN3 --> "0"
            self.pi.write(self.input[3], 1 )                # IN4 --> "1"
            time.sleep(0.010)                                 # Retardo (10 mSeg)

        self.Cleanup_Pins_Motor()                           # Bobinas a "0"

        retnrn True                                         # Función terminada
```

```
def Giro_R_Paso(self, Grados):#.....................................................................# "Giro_Derecha_Paso"
                                                                                                      # [MATRIZ]
    nrpm = self.Conv_Grad_rpm(Grados)                                 # Paso de Grados --> rpm

    for rpm in range (nrpm):
        for i in range (4):

            self.pi.write(self.input[0], self.Paso_R [i][0])          # IN1 --> "X"
            self.pi.write(self.input[1], self.Paso_R [i][1])          # IN2 --> "X"
            self.pi.write(self.input[2], self.Paso_R [i][2])          # IN3 --> "X"
            self.pi.write(self.input[3], self.Paso_R [i][3])          # IN4 --> "X"
            time.sleep(0.010)                                            # Retardo (10 mSeg)

    self.Cleanup_Pins_Motor()                                         # Bobinas a "0"

    return True                                                       # Función terminada

def Giro_L_Paso(self, Grados):#.....................................................................# "Giro_Izquierda_Paso"
                                                                                                      # [MATRIZ]
    nrpm = self.Conv_Grad_rpm(Grados)                                 # Paso de Grados --> rpm

    for rpm in range (nrpm):
        for i in range (4):

            self.pi.write(self.input[0], self.Paso_L [i][0])          # IN1 --> "X"
            self.pi.write(self.input[1], self.Paso_L [i][1])          # IN2 --> "X"
            self.pi.write(self.input[2], self.Paso_L [i][2])          # IN3 --> "X"
            self.pi.write(self.input[3], self.Paso_L [i][3])          # IN4 --> "X"
            time.sleep(0.010)                                         # Retardo (10 mSeg)

    self.Cleanup_Pins_Motor()                                         # Bobinas a "0"

    return True                                                       # Función terminada
```

```
def Giro_R_Paso_C(self, Grados):#...................................................................# "Giro_R_Paso_C"
                                                                                                      # [MATRIZ]
    nrpm = self.Conv_Grad_rpm(Grados)                                 # Paso de Grados --> rpm

    for rpm in range (nrpm):
        for i in range (4):

            self.pi.write(self.input[0], self.Paso_C_R [i][0])        # IN1 --> "X"
            self.pi.write(self.input[1], self.Paso_C_R [i][1])        # IN2 --> "X"
            self.pi.write(self.input[2], self.Paso_C_R [i][2])        # IN3 --> "X"
            self.pi.write(self.input[3], self.Paso_C_R [i][3])        # IN4 --> "X"
            time.sleep(0.010)                                         # Retardo (10 mSeg)

    self.Cleanup_Pins_Motor()                                         # Bobinas a "0"

    return True                                                       # Función terminada

def Giro_L_Paso_C(self, Grados):#...................................................................# "Giro_Izquierda_Paso_Completo"
                                                                                                      # [MATRIZ]
    nrpm = self.Conv_Grad_rpm(Grados)                                 # Paso de Grados --> rpm

    for rpm in range (nrpm):
        for i in range (4):

            self.pi.write(self.input[0], self.Paso_C_L [i][0])        # IN1 --> "X"
            self.pi.write(self.input[1], self.Paso_C_L [i][1])        # IN2 --> "X"
            self.pi.write(self.input[2], self.Paso_C_L [i][2])        # IN3 --> "X"
            self.pi.write(self.input[3], self.Paso_C_L [i][3])        # IN4 --> "X"
            time.sleep(0.010)                                         # Retardo (10 mSeg)

    self.Cleanup_Pins_Motor()                                         # Bobinas a "0"

    return True                                                       # Función terminada
```

```
def Giro_R_MP(self, Grados):#.....................................................................# "Giro_Derecha_Medio_Paso"
                                                                                                    # [MATRIZ]
    nrpm = self.Conv_Grad_rpm(Grados)                                   # Paso de Grados --> rpm

    for rpm in range (nrpm):
        for i in range (8):

            self.pi.write(self.input[0], self.Paso_MP_R [i][0])          # IN1 --> "X"
            self.pi.write(self.input[1], self.Paso_MP_R [i][1])          # IN2 --> "X"
            self.pi.write(self.input[2], self.Paso_MP_R [i][2])          # IN3 --> "X"
            self.pi.write(self.input[3], self.Paso_MP_R [i][3])          # IN4 --> "X"
            time.sleep(0.010)                                            # Retardo (10 mSeg)

    self.Cleanup_Pins_Motor()                                            # Bobinas a "0"

    return True                                                          # Función terminada

def Giro_L_MP(self, Grados):#.....................................................................# "Giro_Izquierda_Medio_Paso"
                                                                                                    # [MATRIZ]
    nrpm = self.Conv_Grad_rpm(Grados)                                   # Paso de Grados --> rpm

    for rpm in range (nrpm):
        for i in range (8):

            self.pi.write(self.input[0], self.Paso_MP_L [i][0])          # IN1 --> "X"
            self.pi.write(self.input[1], self.Paso_MP_L [i][1])          # IN2 --> "X"
            self.pi.write(self.input[2], self.Paso_MP_L [i][2])          # IN3 --> "X"
            self.pi.write(self.input[3], self.Paso_MP_L [i][3])          # IN4 --> "X"
            time.sleep(0.010)                                            # Retardo (10 mSeg)

    self.Cleanup_Pins_Motor()                                            # Bobinas a "0"

    return True                                                          # Función terminada

#************************************OTROS*************************************************************

def Cleanup_Pins_Motor (self):#.....................................................................# "CLEANUP PIN's"
    self.pi.write(self.input[0], 0 )                # IN1 --> "0"                                   # Bobinas a "0"
    self.pi.write(self.input[1], 0 )                # IN2 --> "0"
    self.pi.write(self.input[2], 0 )                # IN3 --> "0"
    self.pi.write(self.input[3], 0 )                # IN4 --> "0"
    time.sleep(0.010)                               # Retardo (10 mSeg)

    return True                                     # Función terminada

def Conv_Grad_rpm(self, nGrados):#.................................................................# "CONVERSIÓN: Grados --> rpm"

    rpm = nGrados * 1.4222

    return round (rpm)
```

CLASE Juego:"Juego_CLASE_v17"

```
import pigpio   # Control GPIO
import time
import random

class Juego:
    def __init__(self, pi, pos_cero,pul_start):                                    # CONSTRUCTOR

        self.pi = pi                # instancia de pigpio

        self.pos_cero  = pos_cero   # Cero Mecánico
        self.pul_start = pul_start  # Pulsador Start

        #.....................................................................# PIN's RASPBERRY

        self.pi.set_mode        (pos_cero,   pigpio.INPUT)      # Tipo IN
        self.pi.set_mode        (pul_start,  pigpio.INPUT)      # Tipo IN
        self.pi.set_pull_up_down(pos_cero,   pigpio.PUD_UP)
        self.pi.set_pull_up_down(pul_start,  pigpio.PUD_UP)
```

```
def Pos_Cero (self):

    while True:
                                                # Llegeix l'estat del polsador
        button_state = self.pi.read(self.pos_cero)

        if button_state == 0:                   # "LOW",  pulsador "SI" presionado
            print("¡REF encontrada!")
            return True

        else:                                   # "HIGH", Pulsador "NO" presionado.
            print("REF no encontrada....")
            return False

        time.sleep(0.1)                         # Retard per evitar sobrecàrrega de la CPU
def Pul_Start (self):

    while True:
                                                # Llegeix l'estat del polsador
        button_state = self.pi.read(self.pul_start)

        if button_state == 0:                   # "LOW",  pulsador "SI" presionado
            print("¡START")
            return True

        else:                                   # "HIGH", Pulsador "NO" presionado.
            print("NO START")
            return False

        time.sleep(0.1)                         # Retard per evitar sobrecàrrega de la CPU
def Check_Cajita (self, distancia,pos_actual,pos_bomba):

    if distancia > 0 and distancia < 10:
        print(f"                    -->  HAY UNA CAJITA  <--")
        if pos_actual == pos_bomba:
            print(f"                        -->  BOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMMM  <--")
            return True


    elif distancia > 10:
        print(f"                    -->  NO HAY NADA  <--"        )
        return False

def Pos_Bomba (self):

    pos_bomba = random.randint(1, 5)
    print(f"La BOMBA esta en la Posición: {pos_bomba}")

    return pos_bomba
```

2.2.3. MULTIPROCESSING

MAIN: "BUSCAMINAS_MAIN_v12"

```
# BUSCAMINAS_MAIN_v12.py"

import pigpio   # Control GPIO
import time
import os

from multiprocessing import Process, Value  # Multiproceso, [Value: variable global para los N_Procesos].

from Motor_pp_CLASE_v12   import *          #  MOTOR
from Distancia_CLASE_v12  import *          #  DISTANCIA



#-------------------------------------------------------------------------------------------------------[DECLARACIÓN VARIABLES]
Final_Ciclo = Value('b', False)

#-------------------------------------------------------------------------------------------------------[PIN's RASPBERRY]
IN  = (4, 5, 6, 27)                                 # MOTOR_1

                                                    # DISTANCIA _X
TRIG_PIN = 23                                       # GPIO23 Activa Detector   (Trigger)
ECHO_PIN = 24                                       # GPIO24 Objeto Detectado  (Echo)

#-------------------------------------------------------------------------------------------------------[OBJETOS]
pi = pigpio.pi()                                    # Conexión al "daemon pigpio"

Motor_1 = Motor(pi, IN)                             # Motor_1

Distancia_1 = SensorDistancia (pi, TRIG_PIN, ECHO_PIN)  # Distancia_1
```

```
#-------------------------------------------------------------------------------------------------------[FUNCIONES]
def Mov_Motor (Motor_1):                                        # MOVIMIENTO MOTOR

    print ("\n.....GIRO DERECHA  : PASO   --> SECUENCIAL....")
    Motor_1.Giro_R          (50)                                # hago 1 Giro a la derecha

    print ("\n.....GIRO IZQUIERDA: PASO   --> SECUENCIAL....")
    Motor_1.Giro_L          (50)                                # hago 1 Giro a la izquierda

    print ("\n.....GIRO DERECHA  : PASO   --> MATRIZ......  ")
    Motor_1.Giro_R_Paso     (100)                               # hago 1 Giro a la derecha

    print ("\n.....GIRO IZQUIERDA: PASO   --> MATRIZ......  ")
    Motor_1.Giro_L_Paso     (100)                               # hago 1 Giro a la izquierda


    print ("\n.....GIRO DERECHA  : PASO_C --> MATRIZ......  ")
    Motor_1.Giro_R_Paso_C   (150)                               # hago 1 Giro a la derecha

    print ("\n.....GIRO IZQUIERDA: PASO_C --> MATRIZ......  ")
    Motor_1.Giro_L_Paso_C   (150)                               # hago 1 Giro a la izquierda


    print ("\n.....GIRO DERECHA  : MP     --> MATRIZ......  ")
    Motor_1.Giro_R_MP       (200)                               # hago 1 Giro a la derecha

    print ("\n.....GIRO IZQUIERDA: MP     --> MATRIZ......  ")
    Motor_1.Giro_L_MP       (200)                               # hago 1 Giro a la izquierda

    Final_Ciclo.value = True                                    # Confirmo que he hecho un ciclo


def Med_Distancia (Distancia_1):                        # MEDIR DISTANCIA

    while Final_Ciclo.value == False:
        dist_x = Distancia_1.mesura_distancia()         # Hago mediciones


        if dist_x > 0:
            #os.system('clear')
            print(f"                                                Distancia Eje X: {dist_x:.2f} cm")  # print ("mensaje")

        time.sleep(1)                                   # Tiempo reposo (1 seg)
```

```
#-------------------------------------------------------------------------------------------[MAIN]
if __name__ == '__main__':


    try:
        while Final_Ciclo.value == False:                                   # Mientras Final_Ciclo == False

            Proceso_1 = Process(target = Mov_Motor,     args=(Motor_1,))    # creo "process_1" (función: "Mov_Motor",    argumento ("Motor_1",))
            Proceso_2 = Process(target = Med_Distancia, args=(Distancia_1,))# creo "process_1" (función: "Med_Distancia",argumento ("Distancia_1",))


            Proceso_1.start()                                               # Inicio Proceso_1 --> (Motor_1)
            Proceso_2.start()                                               # Inicio Proceso_2 --> (Distancia_1)


            Proceso_1.join()                                                # Espero Proceso_1 --> (Motor_1)
            Proceso_2.join()                                                # Espero Proceso_2 --> (Distancia_1)


    #-------------------------------------------------------------------------------------------[INTERRUPCIONES]
    except KeyboardInterrupt:

        print ("\n..... Parando Procesos 1 y 2 .......")

        Motor_1.Cleanup_Pins_Motor ()   # Bobinas a "0"

        Proceso_1.terminate()           # Mata las subrrutina "process_1". Si no, solo mataría al MAIN() --> (Motor_1)
        Proceso_2.terminate()           # Mata las subrrutina "process_2". Si no, solo mataría al MAIN() --> (Distancia_1)

        Proceso_1.join()                # Espera que acabe el terminate() --> (Motor_1)
        Proceso_2.join()                # Espera que acabe el terminate() --> (Distancia_1)

        print ("\n..... Parados Procesos 1 y 2 .......")
        print ("\n")
```

CLASES: *"Motor_pp_CLASE_v12" →Compatible con .v13*

"Distancia_CLASE_v12" ->Compatible con .v13

(Expuestos apartado anterior).

Conclusión:

Secuenciador:

Esta estructura de programación es la más práctica para trabajar, pero nos presenta algunos inconvenientes. Todo debe estar programado en el mismo archivo, y para sistemas en la que se deben controlar varios procesos externos o procesos de gestión, así como en programación extensa se empieza a convertir en un cuello de botella. Ralentizando el funcionamiento del programa.

Multihilos:

De los 3 modos de programación este es el que presenta menos problemas. Es quizás, el más sólido y estable. A cada proceso y gestión externa de información se le asigna un "Hilo" independiente uno de otro, pero en el que se puede intercambiar información entre ellos.

En Multiprocessing se observa que:

Este dispositivo (Raspberry) consta de 4 Nucleos. Que mientras se han utilizado 2 Process, el hardware no presenta problemas. Al incrementar a 3 Process, 3 nucleos, el sistema presenta desbordamiento de memoria.

Es decir, los valores de memoria no corresponden, con los valores de entrada del hardware. En un ciclo inicial, el sistema funciona correctamente, pero a medida que se repinten los ciclos empiezan a mostrarse estos problemas.

Se considera que el cuarto Procesador debe encargarse del funcionamiento de la placa base, así como la sincronización de funcionamiento, intercambio de datos entre de los 3 Procesadores.