TP Python N°2

Objetivos:

Realizar un algoritmo de control para un robot o autómata utilizando una máquina de estado en Python.

- No menos de 4 estados.
- Condicionales que cambien su estado simulando sensores por teclado, por archivo, cuentas matemáticas, tiempo o el método que prefiera.
- Subir el programa a la plataforma indicando nombre de los integrantes del grupo en archivo Python. Con el siguiente formato para el nombre del programa.

  <título del programa>_apellido_TG2.py

1. Introducción

Este trabajo tiene como objetivo implementar una máquina de estados en Python para simular el control de un robot. La propuesta sigue la consigna: mínimo 4 estados y transiciones controladas por condiciones que simulan diferentes tipos de sensores (entrada por teclado, archivo, cuentas matemáticas y tiempo).

El desarrollo se realizó en Google Colab, con el apoyo de ChatGPT y tomando como base el repositorio del profesor.

Link relevantes:

- Colab: colab.research.google.com/drive/1IrvAqaNquXMkkn70FGw4es1GvIWkprtj?usp=sharing
- Repositorio: gitlab.com/nbalich/python_basico_robotica/-/blob/master/maquina_de_estados

2. Descripción del programa

El código se estructura en tres bloques principales, siguiendo el esquema del profesor:

- `setup()`
  - Inicializa el sistema.
  - Define el estado inicial (`E_INICIO`), el nombre del controlador (“Arduino”) y arranca el reloj (`g_clock_inicio`).

- `get_clock()`
  - Calcula el tiempo transcurrido desde el inicio.
  - Se usa como “sensor de tiempo” para decidir cuándo procesar un ciclo de la máquina de estados.

- `loop()`
  - Es el bucle principal de ejecución.
  - En cada ciclo (cada X segundos):
    - Lee diferentes “sensores simulados”:
      - Teclado (`input`) → w, e, c, s, q.
      - Archivo externo (`sensor.txt`) → obstáculos o meta.

- Matemática (función `medir_distancia()`) → genera una distancia aleatoria, si es <25 cm se dispara el estado de evitar.
- Tiempo/batería (función `actualizar_bateria()`) → disminuye con el uso y obliga al robot a entrar en estado de carga.

- Imprime en pantalla el estado actual, valores de sensores y batería.
- Ejecuta una acción asociada al estado.
- Evalúa las condiciones de transición para decidir a qué estado pasar.

3. Estados del sistema

El robot se modela con 5 estados principales:

- INICIO → espera órdenes.
- PATRULLA → se mueve hacia adelante.
- EVITAR → rodea obstáculos.
- CARGAR → entra a estación de carga cuando la batería está baja.
- DETENER → fin de misión.

Las transiciones ocurren según reglas como:

- Teclado: `w` → patrulla, `e` → evitar, `c` → cargar, `s` → detener, `q` → salir.
- Archivo: si `sensor.txt` marca `obstaculo=1`, pasa a EVITAR; si `meta=1`, pasa a DETENER.
- Matemática: si la distancia simulada <25 cm, pasa a EVITAR.
- Tiempo: si la batería ≤20%, pasa a CARGAR; si se recarga al 100%, vuelve a INICIO.

4. Acciones por estado

Cada estado ejecuta una acción simulada, que se imprime por consola:

- INICIO: “Sistema inicializado”.

- PATRULLA: “Patrullando área…”.
- EVITAR: “Evitando obstáculo…”.
- CARGAR: “En estación de carga…”.
- DETENER: “Robot detenido. Fin de misión.”

5. Relación con el repositorio

El código está basado en el archivo original `maquina_de_estados.py` del repositorio gitlab.com/nbalich/python_basico_robotica/-/blob/master/maquina_de_estados/maquina_de_estados.py, adaptando:

- El esquema de `setup()` → `get_clock()` → `loop()`.
- El uso de tiempo como sensor (intervalos de ejecución).
- La idea de avanzar de un estado a otro mediante condicionales.

Sobre esta base, se amplió el modelo:

- Se agregaron sensores simulados extra (archivo, teclado, matemática, batería).
- Se definieron 5 estados en lugar de 3.
- Se imprimen logs detallados de cada ciclo.

6. Código

colab.research.google.com/drive/1IrvAqaNquXMkkn70FGw4es1GvIWkprtj?usp=sharing

```
# -*- coding: UTF-8 -*-

# Paradigmas Tecnologicos II - Robótica

# Trabajo Grupal N°2 - Máquina de Estados (mínimo 4 estados)

# Formato de archivo: <titulo>_apellido_TG2.py

# Realizado en Google Colab con ayuda de ChatGPT
```

```
# Basado en el esquema del Ing. Néstor A. Balich (setup / get_clock / loop)


import time

import os

import random

from datetime import datetime


# ----------------------------

# Estados (códigos y nombres)

# ----------------------------

E_INICIO   = 0

E_PATRULLA = 1

E_EVITAR   = 2

E_CARGAR   = 3

E_DETENER  = 4


NOMBRE_ESTADO = {

    E_INICIO:   "INICIO",

    E_PATRULLA: "PATRULLA",

    E_EVITAR:   "EVITAR",

    E_CARGAR:   "CARGAR",

    E_DETENER:  "DETENER"

}


# ----------------------------

# Variables globales
```

```
# ------------------------------

g_estado = E_INICIO

g_controlador = ""

g_clock_inicio = 0.0

g_bateria = 100                # % batería simulada (tiempo)

g_ultimo_tick = 0              # para imprimir ciclos

g_ciclos = 0


# Archivo de sensores externo (opcional)

ARCH_SENSOR = "sensor.txt"

# Convención de archivo (líneas posibles):

# obstaculo=0|1

# meta=0|1


# ------------------------------

# Setup

# ------------------------------

def setup():

    global g_estado, g_controlador, g_clock_inicio, g_bateria, g_ciclos

    g_estado = E_INICIO

    g_controlador = "Arduino"

    g_clock_inicio = time.time()

    g_bateria = 100

    g_ciclos = 0

    print("=== Máquina de Estados - Robot ===")

       print("Comandos teclado: w=patrullar | e=evitar | c=cargar |
s=detener | q=salir")
```

```
    print("Sensores: teclado / archivo sensor.txt / distancia (cálculo)
/ tiempo (batería)")
    print("-----------------------------------")


# ----------------------------
# Reloj: segundos desde el último intervalo
# ----------------------------
def get_clock():
    global g_clock_inicio
    diferencia = round(time.time() - g_clock_inicio)
    return diferencia % 60


# ----------------------------
# Lectura "sensor" por archivo
# ----------------------------
def leer_sensor_archivo():
    """Lee sensor.txt si existe. Devuelve dict con banderas."""
    data = {"obstaculo": 0, "meta": 0}
    if not os.path.exists(ARCH_SENSOR):
        return data
    try:
        with open(ARCH_SENSOR, "r", encoding="utf-8") as f:
            for linea in f:
                linea = linea.strip().lower()
                if linea.startswith("obstaculo="):
                    data["obstaculo"] = int(linea.split("=", 1)[1])
                elif linea.startswith("meta="):
```

```
                    data["meta"] = int(linea.split("=", 1)[1])
    except Exception as e:
        print(f"[WARN] No se pudo leer {ARCH_SENSOR}: {e}")
    return data


# ----------------------------
# Lectura "sensor" por teclado
# ----------------------------
def leer_teclado_no_obligatorio():
    """
     Pide un comando pero permite Enter para 'ninguno', así no bloquea la lógica.
    """
    try:
        cmd = input("Comando (Enter para ninguno): ").strip().lower()
        return cmd
    except EOFError:
        return ""
    except KeyboardInterrupt:
        return "q"


# ----------------------------
# "Sensor" matemático (distancia simulada)
# ----------------------------
def medir_distancia():
    """
     Simula un sensor de distancia (cm). Usamos una distribución con ruido.
```

```
    """
      # Distancia aleatoria: a veces muy cerca (<25cm) para disparar
EVITAR
    base = random.uniform(10, 120)
    ruido = random.uniform(-5, 5)
    return max(0, base + ruido)


# ----------------------------
# "Sensor" de tiempo (batería)
# ----------------------------
def actualizar_bateria(estado):
    """
    Batería baja con el tiempo y más rápido si patrulla/evita; sube en
carga.
    """
    global g_bateria
    if estado == E_PATRULLA:
        g_bateria -= 2
    elif estado == E_EVITAR:
        g_bateria -= 3
    elif estado == E_INICIO:
        g_bateria -= 1
    elif estado == E_CARGAR:
        g_bateria += 5

    g_bateria = max(0, min(100, g_bateria))

# ----------------------------
```

```
# Acciones por estado (actuadores)
# ------------------------------
def accion_inicio():
    print(">> Sistema inicializado. Esperando orden...")

def accion_patrulla():
    print(">> Patrullando área... (mover adelante)")

def accion_evitar():
    print(">> Evitando obstáculo... (giro y rodeo)")

def accion_cargar():
    print(">> En estación de carga... (recuperando batería)")

def accion_detener():
    print(">> Robot detenido. Fin de misión.")

# ------------------------------
# Transiciones de estado
# ------------------------------
def transiciones(estado, cmd_teclado, sensor_arch, distancia, bateria):
    """
    Reglas de transición:
       - Teclado: w→PATRULLA, e→EVITAR, c→CARGAR, s→DETENER, q→SALIR
(DETENER)
    - Archivo: obstaculo==1 → EVITAR ; meta==1 → DETENER
    - Matemática: distancia < 25cm → EVITAR
```

```
     - Tiempo: batería <= 20% → CARGAR ; batería == 100% y en CARGAR →
INICIO

    - Default: volver a INICIO cuando se completa la acción

    """

    # Salida inmediata por teclado

    if cmd_teclado == "q":

        return E_DETENER



    if cmd_teclado == "w":

        return E_PATRULLA

    if cmd_teclado == "e":

        return E_EVITAR

    if cmd_teclado == "c":

        return E_CARGAR

    if cmd_teclado == "s":

        return E_DETENER



    # Archivo (sensores)

    if sensor_arch.get("meta", 0) == 1:

        return E_DETENER

    if sensor_arch.get("obstaculo", 0) == 1:

        return E_EVITAR



    # Matemática (distancia)

    if distancia < 25:

        return E_EVITAR
```

```
    # Tiempo (batería)
    if bateria <= 20 and estado != E_CARGAR:
        return E_CARGAR
    if estado == E_CARGAR and bateria >= 100:
        return E_INICIO

    # Defaults de flujo
    if estado in (E_PATRULLA, E_EVITAR):
            # tras un ciclo de trabajo vuelve a INICIO si nada fuerza
permanencia
        return E_INICIO

    return estado

# ----------------------------
# Bucle principal (loop)
# ----------------------------
def loop():
    global g_estado, g_clock_inicio, g_ultimo_tick, g_ciclos

    INTERVALO_SEG = 2  # cada 2 segundos procesa un ciclo

    while True:
        # Tick de intervalo estilo ejemplo del profe
        if get_clock() >= INTERVALO_SEG:
            g_clock_inicio = time.time()
            g_ciclos += 1
```

```
            # “Sensores”:
            cmd = leer_teclado_no_obligatorio()         # teclado
            s_arch = leer_sensor_archivo()              # archivo
            dist = medir_distancia()                    # matemático
            actualizar_bateria(g_estado)               # tiempo

            # Log de ciclo
            ahora = datetime.now().strftime("%H:%M:%S")
            print(f"\n--- Ciclo #{g_ciclos} @ {ahora} ---")
                print(f"Estado: {NOMBRE_ESTADO[g_estado]} | Distancia:
{dist:.1f} cm | Batería: {g_bateria}% | Archivo: {s_arch} | Teclado:
'{cmd or '-'}'")

            # Acciones del estado actual
            if g_estado == E_INICIO:
                accion_inicio()
            elif g_estado == E_PATRULLA:
                accion_patrulla()
            elif g_estado == E_EVITAR:
                accion_evitar()
            elif g_estado == E_CARGAR:
                accion_cargar()
            elif g_estado == E_DETENER:
                accion_detener()
                break

            # Transición de estado
```

```
                g_estado = transiciones(g_estado, cmd, s_arch, dist,
g_bateria)



# ----------------------------

# Punto de entrada

# ----------------------------

if __name__ == "__main__":

    setup()

    loop()
```

.