1.Introducción

El presente informe detalla el funcionamiento de un código desarrollado para controlar un robot aspiradora Roomba. Este código está diseñado para proporcionar movimiento, control autónomo y odometría (medición y seguimiento de posición y orientación). A continuación, se describe cada módulo del código, destacando las principales funcionalidades implementadas.

2. Estructura General del Código

El programa está organizado en dos clases principales y una función main para gestionar el flujo principal del programa:

1. **OdometriaRoomba:** Responsable de calcular y mantener la posición y orientación del robot utilizando datos de los encoders de las ruedas.
2. **RoombaController:** Maneja la conexión, control de movimiento y modos de operación de la Roomba.
3. **main():** Gestiona la interacción con el usuario y el ciclo principal de comandos.

Cada sección del código está diseñada para cumplir con un objetivo específico, que se detalla en los apartados siguientes.

2.1. Clase OdometriaRoomba

La clase OdometriaRoomba gestiona la **odometría**, es decir, el cálculo de la posición (x, y) y la orientación angular (theta) del robot a partir de los datos proporcionados por los encoders.

Principales Componentes:

Inicialización (__init__): Se definen parámetros esenciales, como la posición inicial (0,0) y el ángulo inicial 0 rad. También se inicializan los valores de los encoders.

Ajuste de Overflow (ajustar_overflow): Esta función compensa el **desbordamiento** de los encoders cuando superan el valor máximo permitido por el hardware (MAX_TICKS).

Actualización de la Posición (actualizar_posicion): Esta función es uno de los núcleos del sistema de odometría. Calcula la posición y orientación del robot en base a los valores de los encoders. La función permite rastrear la ubicación del robot en tiempo real, fundamental para tareas de navegación autónoma o mapeo del entorno. El flujo de esta función incluye:

- **Cálculo del desplazamiento de las ruedas:**
  Utiliza los valores de ticks (incrementos del encoder) para calcular la distancia recorrida por cada rueda:

  distancia_izq = ticks_izq * MM_POR_TICK
  distancia_der = ticks_der * MM_POR_TICK

- **Cálculo del desplazamiento total y rotación:**
  A partir de la distancia recorrida por ambas ruedas, se calcula el desplazamiento total del robot y el cambio de orientación:

  distancia = (distancia_izq + distancia_der) / 2
  cambio_angular = (distancia_der - distancia_izq) / BASE_RUEDAS

- **Actualización de la posición y el ángulo:**
  Se usa trigonometría para calcular la nueva posición (x, y) del robot considerando el cambio de orientación:

  self.posicion[0] += distancia * cos(self.angulo)
  self.posicion[1] += distancia * sin(self.angulo)
  self.angulo += cambio_angular

- **Impresión de Posición (`imprimir_posicion`):** Muestra los valores calculados en un formato claro para el usuario.

2.2 Clase `RoombaController`

La clase `RoombaController` implementa las funciones de control físico de la Roomba, incluyendo el manejo de comandos y el modo autónomo.

Principales Componentes:
Inicialización y Conexión: esta función establece la conexión con la Roomba utilizando el puerto especificado.
Si la conexión falla, se imprime un mensaje de error y el programa termina.

Funciones de Movimiento: `adelante` y `atras`, que permiten mover la Roomba en línea recta hacia adelante o atrás. **`_mover_roomba`**, que controla el desplazamiento del robot en línea recta. Esta función inicia el movimiento aplicando velocidades en ambas ruebas, monitorea la distancia recorrida usando encores, y detiene el movimiento cuando detecta que se ha alcanzado el objetivo marcado. Además, actualizan los valores de odometría en la clase OdometriaRoomba, asegurando que el sistema tenga un registro preciso del desplazamiento realizado.

Por otra parte tenemos la funcion _girar_roomba que permite girar el robot sobre su eje utilizando velocidades opuestas en las ruedas. Su funcionamiento incluye el cálculo del tiempo necesario para alcanzar el ángulo deseado y considera la

geometría de la base del robot y la velocidad angular. También se actualiza la orientación y esta se registra en `OdometriaRoomba`

Además está la función de **`comportamiento_autonomo`** que implementa un ejemplo de comportamiento autónomo: mover la Roomba en un cuadrado.
for _ in range(4):

```
controlador.adelante(lado)
controlador._girar_roomba(90)
```

2.3 Flujo Principal del Programa (`main`)

La función `main` proporciona un menú interactivo para controlar la Roomba. El usuario puede ingresar comandos, y el programa los interpreta para ejecutar las funciones correspondientes de `RoombaController`. Entre las opciones disponibles están:

Mover en línea recta: `m <distancia en cm> - Mover hacia adelante.`

Girar: `g <ángulo en grados> - Girar la Roomba.`

Modo Autónomo: `auto - Activar modo autónomo.`

3.Manejo de Errores

El código incluye múltiples mecanismos para manejar errores:

Desbordamiento de encoders: Corregido mediante la función `ajustar_overflow`.

Reinicio de encoders: Si los sensores reportan valores inconsistentes, se intenta restablecer la posición.
`self.roomba.SCI.ser.reset_input_buffer()`

Intentos múltiples: Para asegurar lecturas válidas, el programa reintenta obtener datos antes de abandonar una operación.

4.Conclusión

Este código implementa un control robusto y funcional para la Roomba, con las siguientes capacidades destacadas:

1. **Control manual:** Movimientos hacia adelante, atrás y giros precisos.
2. **Modo autónomo:** Navegación predefinida en patrones como un cuadrado.
3. **Odometría:** Seguimiento de posición y orientación con corrección de errores.

La arquitectura modular del programa facilita su mantenimiento y expansión, permitiendo la adición de nuevos modos de operación o mejoras en la precisión del sistema.