Implementación de Controlador en Python para Home Assistant

Para realizar una traducción directa de Simulink a Python, puedes usar **Simulink Coder** para convertir el modelo en código C y luego interactuar con este código en Python mediante bibliotecas como **ctypes** o **cffi**.

Otra opción es usar herramientas como **SymPy** en Python para modelar y resolver ecuaciones simbólicamente si el sistema es complejo.

Para implementar tu controlador en Home Assistant, que es una plataforma de automatización del hogar, puedes seguir un enfoque que integre tu controlador en Python con los servicios de Home Assistant. Aquí te detallo un plan paso a paso.

# 1. Integración de Python con Home Assistant

Home Assistant permite ejecutar código en Python mediante scripts personalizados, automations, o incluso mediante la creación de custom components. Dependiendo de la complejidad de tu controlador, el enfoque puede variar:

- Automations y Scripts: Para lógica sencilla o controladores básicos.

- Custom Component: Para lógica más avanzada y controladores complejos.

# 2. Desarrollo del Controlador en Python

Si ya tienes tu controlador en Python, como lo describimos antes (ej. PID), puedes encapsularlo en un custom component para Home Assistant.

## a) Definir el controlador

Ya que el controlador en Python está desarrollado, el siguiente paso es ajustarlo para ser llamado por Home Assistant.

class PIDController:  
 def \_\_init\_\_(self, kp, ki, kd, setpoint):  
 self.kp = kp  
 self.ki = ki  
 self.kd = kd  
 self.setpoint = setpoint  
 self.integral = 0  
 self.prev\_error = 0  
  
 def update(self, measured\_value, dt):  
 error = self.setpoint - measured\_value  
 self.integral += error \* dt  
 derivative = (error - self.prev\_error) / dt  
 self.prev\_error = error  
 return self.kp \* error + self.ki \* self.integral + self.kd \* derivative

## b) Crear un Custom Component

Home Assistant te permite crear componentes personalizados. Para esto, sigue estos pasos:

1. Estructura del componente:

custom\_components/  
├── my\_controller/  
 ├── \_\_init\_\_.py  
 ├── manifest.json  
 ├── controller.py

2. Configurar `manifest.json`:

{  
 "domain": "my\_controller",  
 "name": "My Custom Controller",  
 "version": "1.0",  
 "dependencies": [],  
 "codeowners": ["@tu\_nombre"],  
 "requirements": []  
}

3. Implementar la lógica en `controller.py`:

import homeassistant.helpers.config\_validation as cv  
from homeassistant.helpers.entity import Entity  
  
def setup\_platform(hass, config, add\_entities, discovery\_info=None):  
 add\_entities([MyControllerEntity()])  
  
class MyControllerEntity(Entity):  
 def \_\_init\_\_(self):  
 self.controller = PIDController(kp=1, ki=0.1, kd=0.05, setpoint=25)  
 self.\_state = None  
  
 def update(self):  
 current\_value = self.hass.states.get("sensor.temperature").state  
 dt = 1  
 self.\_state = self.controller.update(float(current\_value), dt)  
  
 @property  
 def state(self):  
 return self.\_state

4. Configurar `configuration.yaml`:

sensor:  
 - platform: my\_controller

# 3. Comunicación con Sensores y Actuadores

Home Assistant ya está integrado con una variedad de sensores y dispositivos IoT. Puedes utilizar esos sensores como entrada para tu controlador.

# 4. Pruebas y Debugging

Usa los logs de Home Assistant para depurar y ajustar el funcionamiento del controlador.

# 5. Opción de Node-RED (Alternativa)

Si prefieres una interfaz gráfica, puedes usar Node-RED, que está integrado con Home Assistant. Node-RED permite arrastrar y soltar bloques para crear flujos de control.