UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

Máster en Software de Sistemas Distribuidos y Empotrados

Sistemas de Control

Práctica en ROS

Alejandro Casanova Martín

N.º de matrícula: bu0383

Madrid, 2 de enero 2024

Índice

Introducción

En esta práctica se ha implementado el algoritmo genético de la práctica anterior, capaz de ajustar de forma automática un controlador PID, mediante un paquete de ROS.

Desarrollo

Se creó un nuevo paquete llamado *"genetic_tune"* mediante el siguiente comando:

ros2 pkg create genetic_tune --build-type ament_python --dependencies rclpy

En dicho paquete se encuentra el fichero *gen_node.py*, que implementa el nodo de ROS2 correspondiente. El archivo *setup.py* se ha configurado adecuadamente para incluir todos los ficheros necesarios, como puede verse a continuación:

```python
from setuptools import find_packages, setup
from glob import glob

package_name = 'genetic_tune'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        ('share/{}'.format(package_name), glob('launch/*.launch.py')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='robotica',
    maintainer_email='robotica@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'gen_tuner = genetic_tune.gen_node:main'
        ],
    },
)
```

También se creó el fichero *gen_tuner.launcher.py*, cuyo contenido puede verse a continuación:

```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='genetic_tune',  # Reemplaza con el nombre de tu paquete
            executable='gen_tuner',  # Reemplaza con el nombre de tu nodo
            output='screen'         # Puedes elegir 'screen', 'log' o 'both'
        )
    ])

```

Puesta en marcha

Deberán copiarse los ficheros adjuntos en el directorio de trabajo de ROS. Se ha modificado el fichero *utils/Performance.py* del paquete sim*_pkg*, para lograr unos valores parecidos a los obtenidos en la práctica anterior, por lo que deberá sustituirse dicho paquete por el provisto en esta entrega.

A continuación, se generarán todos los ficheros necesarios mediante el comando **build**, definido mediante un alias en el fichero *.bashrc*.

Finalmente, podrán ponerse en marcha ambos nodos ejecutando en diferentes terminales los siguientes comandos:

ros2 launch sim_pkg sim_srv.launch.py

ros2 launch genetic_tune gen_tuner.launch.py

Al finalizar, el nodo del algoritmo genético mostrará por la terminal un resultado parecido a este:

```
[INFO] [gen_tuner-1]: process started with pid [38394]
[gen_tuner-1] [INFO] [1704151129.638649204] [genetic_tuning]:
[gen_tuner-1] Yaml Configuration:
[gen_tuner-1]  Population Size: 200
[gen_tuner-1]  Chromosome Length: 3
[gen_tuner-1]  Generations: 200
[gen_tuner-1]  Mutation Rate: 0.400000
[gen_tuner-1]  Crossover Rate: 0.700000
[gen_tuner-1]  T: 8
[gen_tuner-1]
[gen_tuner-1]  w_ts: 20.000000
[gen_tuner-1]  w_d: 5.000000
[gen_tuner-1]  w_overshoot: 1.000000
[gen_tuner-1]  w_ess: 100.000000
[gen_tuner-1]
[gen_tuner-1] [INFO] [1704151129.639251536] [genetic_tuning]: Generation Progress: 0 / 200
[gen_tuner-1] [INFO] [1704151134.035997616] [genetic_tuning]: Generation Progress: 20 / 200
[gen_tuner-1] [INFO] [1704151138.122619160] [genetic_tuning]: Generation Progress: 40 / 200
[gen_tuner-1] [INFO] [1704151142.355305874] [genetic_tuning]: Generation Progress: 60 / 200
[gen_tuner-1] [INFO] [1704151146.575404613] [genetic_tuning]: Generation Progress: 80 / 200
[gen_tuner-1] [INFO] [1704151150.722030303] [genetic_tuning]: Generation Progress: 100 / 200
[gen_tuner-1] [INFO] [1704151154.889530181] [genetic_tuning]: Generation Progress: 120 / 200
[gen_tuner-1] [INFO] [1704151159.058612955] [genetic_tuning]: Generation Progress: 140 / 200
[gen_tuner-1] [INFO] [1704151163.322429597] [genetic_tuning]: Generation Progress: 160 / 200
[gen_tuner-1] [INFO] [1704151167.702056339] [genetic_tuning]: Generation Progress: 180 / 200
[gen_tuner-1] [INFO] [1704151172.030829548] [genetic_tuning]: Mejor Cromosoma: Kp=0.606928 Ki=0.095548 Kd=0.001428 fitness=421.696362
[gen_tuner-1] [INFO] [1704151172.033246490] [genetic_tuning]:
[gen_tuner-1] Result:
[gen_tuner-1]  Ts: 21.000000
[gen_tuner-1]  d: 0.087509
[gen_tuner-1]  overshoot: 0.104995
[gen_tuner-1]  Ess: 0.011538
[gen_tuner-1]
[INFO] [gen_tuner-1]: process has finished cleanly [pid 38394]
```

Se puede observar que inicialmente se carga y se muestra por terminal la configuración del algoritmo genético. Dichos valores de configuración provienen de un fichero *yaml*. A continuación, se muestra el progreso del algoritmo. Finalmente, se muestra el mejor cromosoma y la respuesta del controlador ajustado.

Ampliación

Se ha incluido el fichero *genetic_tune/genetic_tune/gen_config.yaml* para permitir al usuario la configuración de los distintos parámetros del algoritmo genético, de manera fácil y cómoda.

Adicionalmente, se ha empleado el *logger* para extraer por terminal una traza detallada de los distintos parámetros del algoritmo, su progreso y el resultado final.

Conclusiones

En esta práctica se ha podido comprobar la utilidad de ROS2 para el desarrollo de sistemas de control y aplicaciones de robótica, gracias a su arquitectura distribuida y modular, y a los distintos servicios que facilitan una comunicación eficiente y fluida entre nodos. ROS2 proporciona un nivel de abstracción que facilita considerablemente el desarrollo de aplicaciones, y ofrece robustez ante fallos y escalabilidad.