UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Creación y uso de parámetros de nodos en ROS2

Eugenio Ivorra

En esta práctica, se pretende explorar el uso y las posibilidades que ofrecen la gestión de parámetros de nodos de ROS2. Este conocimiento permitirá configurar y modificar los nodos dependiendo de la aplicación o la situación tanto al iniciar nuevos nodos como para su modificación en tiempo de ejecución. Los objetivos específicos de este módulo son:

- Entender la estructura y funcionamiento de los parámetros en ROS2, incluyendo los diferentes tipos de parámetros y cómo interactuar con ellos.
- Familiarizarse con los callbacks de parámetros y su utilidad en la manipulación dinámica de la configuración en tiempo de ejecución.
- Aprender a usar parámetros en una clase Python con ROS2, permitiendo una gestión más avanzada y estructurada de la configuración.

Índice general

1 Parámetros en ROS 2

1.1 Introducción

Los parámetros en ROS 2 están asociados con nodos individuales. Se utilizan para configurar nodos al inicio (y durante la ejecución), sin cambiar el código. La duración de un parámetro está vinculada a la duración del nodo (aunque el nodo podría implementar algún tipo de persistencia para recargar valores después de reiniciar).

Los parámetros son identificados por el nombre del nodo, el espacio de nombres del nodo, el nombre del parámetro y el espacio de nombres del parámetro. Proporcionar un espacio de nombres para el parámetro es opcional.

1.2 Tipos de Parámetros

Cada parámetro consta de una clave, un valor y un descriptor. La clave es una cadena y el valor es uno de los siguientes tipos: bool, int64, float64, string, byte[], bool[], int64[], float64[] o string[]. Por defecto, todos los descriptores están vacíos, pero pueden contener descripciones de parámetros, rangos de valores, información de tipo y restricciones adicionales.

Por defecto, un nodo necesita declarar todos los parámetros que aceptará durante su vida útil. Esto es para que el tipo y el nombre de los parámetros estén bien definidos al iniciar el nodo, lo que reduce las posibilidades de una configuración incorrecta más adelante.

Para algunos tipos de nodos, no se conocerán todos los parámetros de antemano. En estos casos, el nodo puede ser instanciado con configurar `allow_undeclared_parameters=True` cuando crees una instancia del nodo, lo que permitirá obtener y establecer parámetros en el nodo incluso si no han sido declarados.

Cada parámetro en un nodo ROS 2 tiene uno de los tipos de parámetros predefinidos mencionado anteriormente. Por defecto, los intentos de cambiar el tipo de un parámetro declarado en tiempo de ejecución fallarán. Esto evita errores comunes, como introducir un valor booleano en un parámetro entero.

Si un parámetro necesita ser de varios tipos diferentes, y el código que utiliza el parámetro puede manejarlo, este comportamiento predeterminado puede ser cambiado. Cuando se declara el parámetro, debe ser declarado usando un *ParameterDescriptor* con la variable de miembro *dynamic_ typing* configurada como verdadero.

1.3 Callbacks de Parámetros

Un nodo en ROS 2 escrito en Python ofrece la posibilidad de registrar callbacks para ser notificado de cambios en sus parámetros. Existen principalmente dos tipos de callbacks relacionados con parámetros, y ambos son opcionales.

1.3.1 Callback de Establecimiento de Parámetros

Este callback es invocado cuando se intenta establecer o modificar un parámetro. Se puede registrar a través de la función `add_on_set_parameters_callback` proporcionada por la API del nodo. Cuando se invoca, este callback recibe una lista de objetos `Parameter`, que son inmutables. Tras procesar estos objetos, debe retornar un `rcl_interfaces/msg/SetParametersResult`. La principal utilidad de este callback es permitir al programador inspeccionar y, si se desea, rechazar un cambio propuesto en un parámetro.

```
from rcl_interfaces.msg import SetParametersResult

def on_set_parameters_callback(self, params):
    for param in params:
        # Por ejemplo, rechazar cualquier intento de cambiar el parámetro
        ↪  "critical_parameter"
        if param.name == 'critical_parameter':
            return SetParametersResult(successful=False, reason='El parámetro
            ↪  "critical_parameter" no puede ser modificado.')

    # Si no hay problemas, aceptar los cambios
    return SetParametersResult(successful=True)
```

En este ejemplo, el callback rechaza cualquier intento de cambiar un parámetro llamado `critical_parameter`. Por supuesto, este es solo un ejemplo, y en un caso real, la lógica dentro del callback dependería de las necesidades específicas del programa.

1.3.2 Callback en Evento de Parámetros

El segundo tipo de callback se activa en respuesta a eventos de parámetros. Para registrarlo, se utiliza la función `on_parameter_event` disponible en las APIs del cliente de parámetros. A diferencia del callback anterior, este recibe un objeto `rcl_interfaces/msg/ParameterEvent` y no tiene valor de retorno. Se ejecuta después de que todos los parámetros implicados en el evento hayan sido exitosamente establecidos, modificados o eliminados. Su objetivo principal es brindar la capacidad de responder a cambios que ya han sido efectuados sobre los parámetros.

```
def on_parameter_event_callback(self, parameter_event):
    # Lógica para responder al evento
    for new_param in parameter_event.new_parameters:
        self.get_logger().info(f"Nuevo parámetro: {new_param.name} =
        ↪  {new_param.value}")

    for changed_param in parameter_event.changed_parameters:
```

```
            self.get_logger().info(f"Parámetro modificado: {changed_param.name} =
            ↪  {changed_param.value}")

        for deleted_param in parameter_event.deleted_parameters:
            self.get_logger().info(f"Parámetro eliminado: {deleted_param.name}")
```

Este ejemplo imprime información sobre nuevos parámetros, parámetros modificados y parámetros eliminados cuando se activa el callback.

1.4 Interactuando con los Parámetros

Los nodos de ROS2 pueden realizar operaciones en los parámetros a través de las APIs de nodos como veremos en una sección posterior. Los procesos externos también pueden realizar operaciones de parámetros a través de servicios de parámetros que se crean por defecto cuando se instancia un nodo. Los servicios que se crean por defecto son:

- /node_name/describe_parameters:
  Usa un tipo de servicio de *rcl_interfaces/srv/DescribeParameters*. Dada una lista de nombres de parámetros, devuelve una lista de descriptores asociados a los parámetros.

- /node_name/get_parameter_types:
  Usa un tipo de servicio de *rcl_interfaces/srv/GetParameterTypes*. Dada una lista de nombres de parámetros, devuelve una lista de tipos de parámetros asociados a los parámetros.

- /node_name/get_parameters:
  Usa un tipo de servicio de *rcl_interfaces/srv/GetParameters*. Dada una lista de nombres de parámetros, devuelve una lista de valores de parámetros asociados a los parámetros.

- /node_name/list_parameters:
  Usa un tipo de servicio de `rcl_interfaces/srv/ListParameters`. Dada una lista opcional de prefijos de parámetros, devuelve una lista de los parámetros disponibles con ese prefijo. Si los prefijos están vacíos, devuelve todos los parámetros.

- /node_name/set_parameters:
  Usa un tipo de servicio de `rcl_interfaces/srv/SetParameters`. Dada una lista de nombres y valores de parámetros, intenta establecer los parámetros en el nodo. Devuelve una lista de resultados al intentar establecer cada parámetro; algunos pueden haber tenido éxito y otros no.

- /node_name/set_parameters_atomically:
  Usa un tipo de servicio de `rcl_interfaces/srv/SetParametersAtomically`. Dada una lista de nombres y valores de parámetros, intenta establecer los parámetros en el nodo. Devuelve un único resultado al intentar establecer todos los parámetros, así que si uno falla, todos fallan.

1.4.1 Estableciendo valores de parámetros iniciales al ejecutar un nodo

Al lanzar un nodo en ROS 2, es posible definir valores iniciales para sus parámetros. Estos valores pueden ser especificados de dos maneras principales:

1. **A través de Argumentos en Línea de Comandos:** Es posible definir directamente en la línea de comandos cada parámetro y su valor correspondiente. Este método es útil para la modificación rápida de algunos parámetros sin la necesidad de editar archivos externos.

```
ros2 run paquete_nombre nodo_nombre --ros-args -p parametro1:=valor1
↪ -p parametro2:=valor2
```

2. **Usando Archivos YAML:** Si se tiene una configuración de parámetros más compleja o se desea mantener una estructura más organizada y reutilizable, se pueden utilizar archivos YAML. Una vez que el archivo está creado, se puede pasar al nodo al momento de ejecutarlo.

```
ros2 run paquete_nombre nodo_nombre --ros-args --params-file
↪ ruta/al/archivo.yaml
```

   Aquí se muestra un ejemplo de Archivo YAML para parámetros. Esta archivo tiene que tener una estructura específica. A continuación, se muestra un ejemplo de cómo podría verse dicho archivo:

```yaml
nodo_nombre:
  ros__parameters:
    parametro1: valor1
    parametro2: valor2
    parametro3:
      subparametro1: subvalor1
      subparametro2: subvalor2
    parametro_bool: true
    parametro_array: [1, 2, 3, 4, 5]
```

   Dónde:

   - `nodo_nombre`: Corresponde al nombre del nodo para el cual se están definiendo los parámetros. Debe ser sustituido por el nombre real del nodo en cuestión.
   - `ros__parameters`: Es una clave especial que señala que las siguientes entradas son parámetros para el nodo.
   - Las entradas subsiguientes (`parametro1`, `parametro2`, etc.) representan las claves y valores de los parámetros. En este ejemplo, se incluyen parámetros con valores simples, un parámetro estructurado como un diccionario (con subparámetros), un parámetro de tipo booleano y uno que es una lista.

Elegir uno u otro método dependerá de las necesidades específicas y de la complejidad del nodo en cuestión

1.4.2 Usando la herramienta de línea de comandos ros2 param

Los parámetros en ROS 2 pueden obtenerse, establecerse, listarse y describirse a través de un conjunto de servicios. La herramienta de línea de comandos *ros2 param* es un envoltorio alrededor de estas llamadas de servicio que facilita la manipulación de parámetros desde la línea de comandos. A continuación se describe cada uno de estos comandos junto con ejemplos:

`delete` Este comando se utiliza para eliminar un parámetro específico.

```
ros2 param delete /nombre_del_nodo parametro_a_eliminar
```

`describe` Muestra información descriptiva sobre los parámetros declarados.

```
ros2 param describe /nombre_del_nodo parametro_a_describir
```

`dump` Permite exportar los parámetros de un nodo a un archivo en formato YAML.

```
ros2 param dump /nombre_del_nodo -f path/del/archivo.yaml
```

`get` Obtiene el valor de un parámetro específico.

```
ros2 param get /nombre_del_nodo parametro_solicitado
```

`list` Produce una lista de los parámetros disponibles.

```
ros2 param list /nombre_del_nodo
```

`load` Carga un archivo de parámetros en un nodo específico.

```
ros2 param load /nombre_del_nodo path/del/archivo.yaml
```

`set` Establece el valor de un parámetro.

```
ros2 param set /nombre_del_nodo nombre_del_parametro nuevo_valor
```

Para hacer uso de cualquiera de estos comandos, uno generalmente precede el comando con `ros2 param`. En los ejemplos anteriores, `/nombre_del_nodo` representa el nombre del nodo ROS2 con el que se desea interactuar, y deberá ser reemplazado por el nombre real del nodo en cuestión.

1.4.3 Modificación de parámetros usando rqt

rqt es una interfaz gráfica de usuario en ROS que proporciona una forma visual y más amigable para interactuar con varios aspectos del sistema ROS, incluida la gestión de parámetros. La herramienta específica dentro de *rqt* para trabajar con parámetros es *rqt_ reconfigure*.

A continuación, se muestra cómo se pueden modificar los parámetros usando *rqt*:

1. **Lance su nodo ROS (o sistema):**
   Asegúrese de que su nodo ROS esté en ejecución y que los parámetros que desea modificar estén activos y hayan sido declarados.

2. **Inicie rqt:**
   En una terminal, ejecute:

   ```
   rqt
   ```

3. **Elija el complemento rqt_reconfigure:**
   Una vez que *rqt* esté en marcha, vaya al menú *Plugins* y busque *Configuration* o *Reconfigure*. Seleccione la opción que dice *Dynamic Reconfigure* o algo similar. Esto debería abrir el complemento *rqt_ reconfigure* dentro de la interfaz de *rqt*.

4. **Seleccione su nodo:**
   En el panel izquierdo de *rqt_ reconfigure*, verá una lista de todos los nodos que tienen parámetros dinámicos configurables. Haga clic en el nodo cuyos parámetros desea modificar.

5. **Modifique los parámetros:**
   Una vez que haya seleccionado un nodo, en el panel derecho aparecerán todos los parámetros dinámicos de ese nodo. Puede cambiar estos parámetros usando los controles deslizantes, casillas de verificación, campos de texto, etc., según el tipo de parámetro.

6. **Confirme y aplique los cambios:**
   A medida que modifica los parámetros, los cambios se reflejarán en tiempo real en el nodo ROS en ejecución. No es necesario un paso de "aplicación" adicional, ya que *rqt_ reconfigure* actualizará los parámetros en tiempo real.

1.4.4 Actividad

Averigua los parámetros que puedes modificar en turtlesim y modifica los colores del fondo del mismo.

1.5 Uso de parámetros en una clase Python con ROS2

En el desarrollo con ROS2, a menudo se necesita crear nodos que tengan parámetros configurables. Estos parámetros permiten que el comportamiento del nodo sea ajustable sin tener que modificar y recompilar el código. Esta sección proporciona una guía detallada sobre cómo definir, usar y modificar parámetros en una clase Python para ROS2.

1.5.1 Creación del nodo Python

A continuación, vamos a definir nuestro nodo en Python:

1. **Escribir el código del nodo**:
   En el directorio `ros2_ws/src/mi_paquete_python/mi_paquete_python/`, cree un archivo llamado `python_parameters_node.py` y pegue el código proporcionado:

```python
import rclpy
import rclpy.node

class MinimalParam(rclpy.node.Node):
    def __init__(self):
        super().__init__('minimal_param_node')
        self.declare_parameter('my_parameter', 'world')
        self.timer = self.create_timer(1, self.timer_callback)

    def timer_callback(self):
        my_param =
        ↪  self.get_parameter('my_parameter').get_parameter_value()
            .string_value
        self.get_logger().info('Hello %s!' % my_param)
        my_new_param = rclpy.parameter.Parameter(
            'my_parameter',
            rclpy.Parameter.Type.STRING,
            'world'
        )
        all_new_parameters = [my_new_param]
        self.set_parameters(all_new_parameters)

def main():
    rclpy.init()
    node = MinimalParam()
    rclpy.spin(node)

if __name__ == '__main__':
    main()
```

2. **Análisis del código**: Las declaraciones `import` al inicio son las dependencias requeridas. La clase `MinimalParam` define un nodo ROS2 que tiene un parámetro llamado `my_parameter`. Este parámetro se declara y se utiliza en el callback del temporizador.

3. **(Opcional) Uso de ParameterDescriptor**: Si desea añadir metadatos adicionales o restricciones a su parámetro, puede usar un `ParameterDescriptor` como se muestra en el código siguiente:

```python
# ...

class MinimalParam(rclpy.node.Node):
    def __init__(self):
        super().__init__('minimal_param_node')
```

```
            from rcl_interfaces.msg import ParameterDescriptor
            my_parameter_descriptor =
            ↪  ParameterDescriptor(description='¡Este parámetro es
            ↪  mío!')

            self.declare_parameter('my_parameter', 'world',
            ↪  my_parameter_descriptor)
            self.timer = self.create_timer(1, self.timer_callback)
```

4. **Configuración de puntos de entrada**: Esto asegura que su script Python sea ejecutable como un nodo ROS2. Modifique `setup.py` para agregar el punto de entrada necesario:

```
    entry_points={
        'console_scripts': [
            'minimal_param_node =
            ↪  mi_paquete_python.python_parameters_node:main',
        ],
    },
```

1.5.2 Compilación y prueba

Con nuestro nodo definido, procedamos a compilar y probarlo:

1. **Compilación**: Asegúrese de estar en `ros2_ws` y ejecute:

```
    colcon build --packages-select  mi_paquete_python
```

2. **Ejecución**: Inicie una nueva terminal, navegue a `ros2_ws`, cargue el entorno y ejecute el nodo:

```
    source install/setup.bash
    ros2 run  mi_paquete_python minimal_param_node
```

   La terminal debería devolver el siguiente mensaje cada segundo:

```
    [INFO] [parameter_node]: ¡Hello world!
```

3. **Modificación de parámetros**: Para ver y modificar parámetros en tiempo real mientras el nodo está en ejecución, use los comandos de `ros2 param`. Por ejemplo:

```
    ros2 param set /minimal_param_node my_parameter profesor
```

   Sabrás que salió bien si obtienes la salida `Establecer parámetro exitosamente`. Si miras la otra terminal, deberías ver que la salida cambia a `[INFO] [minimal_param_node]: ¡Hello profesor!`. Luego volverá al valor por defecto puesto que el nodo lo vuelve a sobreescribir.

1.6 Actividad: Establecimiento de Zonas Seguras Circulares en Turtlesim usando Parámetros de ROS2

Objetivo: Utilizar parámetros de ROS2 en Python para definir una zona segura circular en el espacio de Turtlesim, y evitar que la tortuga ingrese a esta área.

1.6.1 Descripción:

1. **Inicialización y Configuración:**

   - Lanza el simulador Turtlesim.
   - Crea un nodo Python llamado `safe_zone_controller` que suscriba al tópico `/turtle1/pose` para recibir la posición actual de la tortuga.

2. **Implementación de Parámetros:**

   - Define tres parámetros en el nodo `safe_zone_controller`: `center_x`, `center_y` y `radius`. Estos definirán una zona circular en la que la tortuga no debe ingresar.
   - Establece valores iniciales para estos parámetros, por ejemplo, un círculo en el centro del espacio de Turtlesim con un radio determinado de uno.

3. **Lógica del Nodo:**

   - El nodo debe monitorear continuamente la posición de la tortuga con una frecuencia de un segundo.
   - Si la tortuga se aproxima al borde del círculo definido (es decir, la distancia desde la tortuga al centro es cercana al radio), el nodo deberá publicar un mensaje en el tópico `/turtle1/cmd_vel` para detener su movimiento y así evitar que ingrese a la zona prohibida.
   - Si la tortuga logra ingresar al círculo (a pesar de las contramedidas), el nodo debe imprimir una alerta en la consola.
   - Añade un callback en evento de parámetros para notificar cuándo una zona ha sido modificada.

4. **Control Dinámico:**

   - Mientras el nodo está en ejecución, usa la herramienta `ros2 param set` para modificar los valores de `center_x`, `center_y` y `radius` y así redefinir la zona segura circular en tiempo real.
   - En una terminal separada, lanza la herramienta `teleop` para mover manualmente la tortuga en el espacio de Turtlesim y comprueba que reacciona a la zona segura.

Bibliografía

Web de los tutoriales de ROS (2023). https://docs.ros.org/en/humble/Tutorials.html.

Web de ROS2 Humble (2023). https://docs.ros.org/en/humble/index.html.