UTS ROBOTIKA

Disusun untuk memenuhi tugas Mata Kuliah Robotika

Disusun oleh:

Muhammad Faisal Ramadhan (1103203227)

PROGRAM SARJANA TEKNIK KOMPUTER

FAKULTAS TEKNIK ELEKTRO

TELKOM UNIVERSITY

2022/2023

Python code:

```
"""Launch Webots and the controller."""


import os
import pathlib
import launch
from launch.substitutions import LaunchConfiguration
from launch.actions import DeclareLaunchArgument
from launch.substitutions.path_join_substitution import PathJoinSubstitution
from launch import LaunchDescription
from launch_ros.actions import Node
from ament_index_python.packages import get_package_share_directory, get_packages_with_prefixes
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.actions import IncludeLaunchDescription
from webots_ros2_driver.webots_launcher import WebotsLauncher
from webots_ros2_driver.utils import controller_url_prefix



def get_ros2_nodes(*args):
    optional_nodes = []
    package_dir = get_package_share_directory('webots_ros2_tiago')
    use_rviz = LaunchConfiguration('rviz', default=False)
    use_nav = LaunchConfiguration('nav', default=False)
    use_slam_toolbox = LaunchConfiguration('slam_toolbox', default=False)
    use_slam_cartographer = LaunchConfiguration('slam_cartographer', default=False)
    robot_description = pathlib.Path(os.path.join(package_dir, 'resource', 'tiago_webots.urdf')).read_text()
    ros2_control_params = os.path.join(package_dir, 'resource', 'ros2_control.yml')
```

```
nav2_params = os.path.join(package_dir, 'resource', 'nav2_params.yaml')
toolbox_params = os.path.join(package_dir, 'resource',
                 'slam_toolbox_params.yaml')
nav2_map = os.path.join(package_dir, 'resource', 'map.yaml')
cartographer_config_dir = os.path.join(package_dir, 'resource')
cartographer_config_basename = 'cartographer.lua'
use_sim_time = LaunchConfiguration('use_sim_time', default=True)

controller_manager_timeout = ['--controller-manager-timeout', '500']
controller_manager_prefix = 'python.exe' if os.name == 'nt' else ''

use_deprecated_spawner_py = 'ROS_DISTRO' in os.environ and
                            os.environ['ROS_DISTRO'] == 'foxy'

diffdrive_controller_spawner = Node(
    package='controller_manager',
    executable='spawner' if not use_deprecated_spawner_py else 'spawner.py',
    output='screen',
    prefix=controller_manager_prefix,
    arguments=['diffdrive_controller'] + controller_manager_timeout,
)

joint_state_broadcaster_spawner = Node(
    package='controller_manager',
    executable='spawner' if not use_deprecated_spawner_py else 'spawner.py',
    output='screen',
    prefix=controller_manager_prefix,
    arguments=['joint_state_broadcaster'] + controller_manager_timeout,
)
```

```
mappings = [('/diffdrive_controller/cmd_vel_unstamped', '/cmd_vel')]
if 'ROS_DISTRO' in os.environ and os.environ['ROS_DISTRO'] in ['humble',
                'rolling']:
    mappings.append(('/diffdrive_controller/odom', '/odom'))


tiago_driver = Node(
    package='webots_ros2_driver',
    executable='driver',
    output='screen',
    additional_env={'WEBOTS_CONTROLLER_URL':  controller_url_prefix()
                + 'Tiago_Iron'},
    parameters=[
        {'robot_description': robot_description,
         'use_sim_time': use_sim_time,
         'set_robot_state_publisher': True},
        ros2_control_params
    ],
    remappings=mappings
)


robot_state_publisher = Node(
    package='robot_state_publisher',
    executable='robot_state_publisher',
    output='screen',
    parameters=[{
        'robot_description': '<robot name=""><link name=""/></robot>'
    }],
)


footprint_publisher = Node(
```

```
    package='tf2_ros',
    executable='static_transform_publisher',
    output='screen',
    arguments=['0', '0', '0', '0', '0', '0', 'base_link', 'base_footprint'],
)

rviz_config = os.path.join(get_package_share_directory('webots_ros2_tiago'),
                           'resource', 'default.rviz')
rviz = Node(
    package='rviz2',
    executable='rviz2',
    output='screen',
    arguments=['--display-config=' + rviz_config],
    parameters=[{'use_sim_time': use_sim_time}],
    condition=launch.conditions.IfCondition(use_rviz)
)

# Navigation
if 'nav2_bringup' in get_packages_with_prefixes():
    optional_nodes.append(IncludeLaunchDescription(
        PythonLaunchDescriptionSource(os.path.join(
            get_package_share_directory('nav2_bringup'), 'launch',
                'bringup_launch.py')),
        launch_arguments=[
            ('map', nav2_map),
            ('params_file', nav2_params),
            ('use_sim_time', use_sim_time),
        ],
        condition=launch.conditions.IfCondition(use_nav)))
```

```
# SLAM
cartographer = Node(
    package='cartographer_ros',
    executable='cartographer_node',
    name='cartographer_node',
    output='screen',
    parameters=[{'use_sim_time': use_sim_time}],
    arguments=['-configuration_directory', cartographer_config_dir,
               '-configuration_basename', cartographer_config_basename],
    condition=launch.conditions.IfCondition(use_slam_cartographer))
optional_nodes.append(cartographer)

if 'ROS_DISTRO' in os.environ and os.environ['ROS_DISTRO'] == 'foxy':
    grid_executable = 'occupancy_grid_node'
else:
    grid_executable = 'cartographer_occupancy_grid_node'
cartographer_grid = Node(
    package='cartographer_ros',
    executable=grid_executable,
    name='cartographer_occupancy_grid_node',
    output='screen',
    parameters=[{'use_sim_time': use_sim_time}],
    arguments=['-resolution', '0.05'],
    condition=launch.conditions.IfCondition(use_slam_cartographer))
optional_nodes.append(cartographer_grid)

slam_toolbox = Node(
    parameters=[toolbox_params,
                {'use_sim_time': use_sim_time}],
```

```
            package='slam_toolbox',
            executable='async_slam_toolbox_node',
            name='slam_toolbox',
            output='screen',
            condition=launch.conditions.IfCondition(use_slam_toolbox)
        )
        optional_nodes.append(slam_toolbox)

        # Wait for the simulation to be ready to start RViz and the navigation
        nav_handler = launch.actions.RegisterEventHandler(
            event_handler=launch.event_handlers.OnProcessExit(
                target_action=diffdrive_controller_spawner,
                on_exit=[rviz] + optional_nodes
            )
        )

        return [
            joint_state_broadcaster_spawner,
            diffdrive_controller_spawner,
            nav_handler,
            robot_state_publisher,
            tiago_driver,
            footprint_publisher,
        ]


    def generate_launch_description():
        package_dir = get_package_share_directory('webots_ros2_tiago')
        world = LaunchConfiguration('world')
```

```
mode = LaunchConfiguration('mode')

webots = WebotsLauncher(
    world=PathJoinSubstitution([package_dir, 'worlds', world]),
    mode=mode,
    ros2_supervisor=True
)

# The following line is important!
# This event handler respawns the ROS 2 nodes on simulation reset (supervisor
                process ends).
reset_handler = launch.actions.RegisterEventHandler(
    event_handler=launch.event_handlers.OnProcessExit(
        target_action=webots._supervisor,
        on_exit=get_ros2_nodes,
    )
)

return LaunchDescription([
    DeclareLaunchArgument(
        'world',
        default_value='default.wbt',
        description='Choose one of the world files from
                `/webots_ros2_tiago/world` directory'
    ),
    DeclareLaunchArgument(
        'mode',
        default_value='realtime',
        description='Webots startup mode'
    ),
```

```
    webots,
    webots._supervisor,

    # This action will kill all nodes once the Webots simulation has exited
    launch.actions.RegisterEventHandler(
        event_handler=launch.event_handlers.OnProcessExit(
            target_action=webots,
            on_exit=[
                launch.actions.UnregisterEventHandler(
                    event_handler=reset_handler.event_handler
                ),
                launch.actions.EmitEvent(event=launch.events.Shutdown())
            ],
        )
    ),

    # Add the reset event handler
    reset_handler
] + get_ros2_nodes())
```

Kode tersebut merupakan program untuk meluncurkan Webots dan kontroler pada ROS 2 (Robot Operating System 2). Program ini menggunakan beberapa package dan node di ROS 2 untuk mengontrol robot simulasi pada lingkungan Webots.

Program tersebut mengimport beberapa package seperti os, pathlib, launch, launch_ros, dan lain-lain, serta beberapa module yang terdapat di package tersebut. Kemudian program mengatur beberapa konfigurasi dan argumen launch, seperti path file konfigurasi, url controller, parameter untuk node, dan lain-lain. Selanjutnya program membuat beberapa node di ROS 2, seperti tiago_driver, robot_state_publisher, footprint_publisher, dan lain-lain.

Program tersebut juga menggunakan beberapa kondisi dan argumen untuk memilih node yang akan dijalankan, seperti argumen use_rviz, use_nav, use_slam_toolbox, dan use_slam_cartographer yang masing-masing menentukan apakah node rviz, navigation, atau slam akan dijalankan.

Kode tersebut cukup kompleks dan menunjukkan adanya integrasi antara lingkungan Webots dan ROS 2.