Solution Actions Quiz

Solution Quiz

For the **actions_quiz_msg** package:

In [ ]:
```
int32 seconds
---
bool status
float64 total_dist
---
float64 current_dist
```

In [ ]:
```
cmake_minimum_required(VERSION 3.5)
project(actions_quiz_msg)

# Default to C99
if(NOT CMAKE_C_STANDARD)
  set(CMAKE_C_STANDARD 99)
endif()

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(action_msgs REQUIRED)
find_package(rosidl_default_generators REQUIRED)
# uncomment the following section in order to fill in
# further dependencies manually.
# find_package(<dependency> REQUIRED)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # uncomment the line when a copyright and license is not present in all source files
  #set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # uncomment the line when this package is not in a git repo
  #set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

rosidl_generate_interfaces(${PROJECT_NAME}
  "action/Distance.action"
)

ament_package()
```

For the **actions_quiz** package:

In [ ]:
```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='actions_quiz',
            executable='actions_quiz_server',
            name='actions_quiz_server',
            output='screen'),
    ])
```

In [ ]:

```
import time
import math

import rclpy
from rclpy.action import ActionServer
from rclpy.node import Node

from rclpy.executors import MultiThreadedExecutor
from rclpy.callback_groups import MutuallyExclusiveCallbackGroup, ReentrantCallbackGroup

from actions_quiz_msg.action import Distance

from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
from std_msgs.msg import Float32
from rclpy.qos import ReliabilityPolicy, QoSProfile


class MyActionServer(Node):

    def __init__(self):
        super().__init__('quiz_action_server')
        self.action_callback_group = ReentrantCallbackGroup()
        self._action_server = ActionServer(
            self, Distance, 'distance_as', self.execute_callback, callback_group=self.action_callback_group)
        self.cmd = Twist()
        self.publisher_ = self.create_publisher(Float32, 'total_distance', 10)
        self.odom_callback_group = self.action_callback_group
        self.odom_subscriber = self.create_subscription(
            Odometry,
            '/odom',
            self.odom_callback,
            QoSProfile(depth=10, reliability=ReliabilityPolicy.BEST_EFFORT),
            callback_group=self.odom_callback_group)  # is the most used to read LaserScan data and some sensor data.

    def odom_callback(self, msg):
        self.pos = [msg.pose.pose.position.x, msg.pose.pose.position.y]

    def dist_diff(self, a, b):
        return math.sqrt((a[0]-b[0])**2+(a[1]-b[1])**2)

    def execute_callback(self, goal_handle):

        self.get_logger().info('Executing goal...')

        feedback_msg = Distance.Feedback()
        topic_msg = Float32()
        feedback_msg.current_dist = 0.0
        last_pos = self.pos

        for i in range(1, goal_handle.request.seconds):
            feedback_msg.current_dist = feedback_msg.current_dist + \
                self.dist_diff(last_pos, self.pos)
            last_pos = self.pos
            self.get_logger().info('Feedback: %s ' % str(feedback_msg))

            goal_handle.publish_feedback(feedback_msg)
            topic_msg.data = feedback_msg.current_dist
            self.publisher_.publish(topic_msg)
            time.sleep(1)

        goal_handle.succeed()

        feedback_msg.current_dist = feedback_msg.current_dist + \
            self.dist_diff(last_pos, self.pos)
        result = Distance.Result()
        result.status = True
        result.total_dist = feedback_msg.current_dist
        self.get_logger().info('Result: {0}'.format(result.status))
        return result


def main(args=None):
    rclpy.init(args=args)

    my_action_server = MyActionServer()
    executor = MultiThreadedExecutor()
    executor.add_node(my_action_server)

    executor.spin()


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

In [ ]:
```
from launch import LaunchDescription
from launch_ros.actions import Node


def generate_launch_description():
    return LaunchDescription([
        Node(
            package='actions_quiz',
            executable='actions_quiz_client',
            name='actions_quiz_client',
            output='screen'),
    ])
```

In [ ]:
```
import rclpy
from rclpy.action import ActionClient
from rclpy.node import Node
from actions_quiz_msg.action import Distance


class MyActionClient(Node):

    def __init__(self):
        super().__init__('my_action_client')
        self._action_client = ActionClient(self, Distance, 'distance_as')

    def send_goal(self, seconds):
        goal_msg = Distance.Goal()
        goal_msg.seconds = seconds

        self._action_client.wait_for_server()
        self._send_goal_future = self._action_client.send_goal_async(
            goal_msg, feedback_callback=self.feedback_callback)

        self._send_goal_future.add_done_callback(self.goal_response_callback)

    def goal_response_callback(self, future):
        goal_handle = future.result()
        if not goal_handle.accepted:
            self.get_logger().info('Goal rejected :(')
            return

        self.get_logger().info('Goal accepted :)')

        self._get_result_future = goal_handle.get_result_async()
        self._get_result_future.add_done_callback(self.get_result_callback)

    def get_result_callback(self, future):
        result = future.result()
        self.get_logger().info('Result: %s' % str(result))
        rclpy.shutdown()

    def feedback_callback(self, feedback_msg):
        feedback = feedback_msg
        self.get_logger().info('Received feedback: %s' % str(feedback))


def main(args=None):
    rclpy.init(args=args)

    action_client = MyActionClient()

    action_client.send_goal(20)

    rclpy.spin(action_client)


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

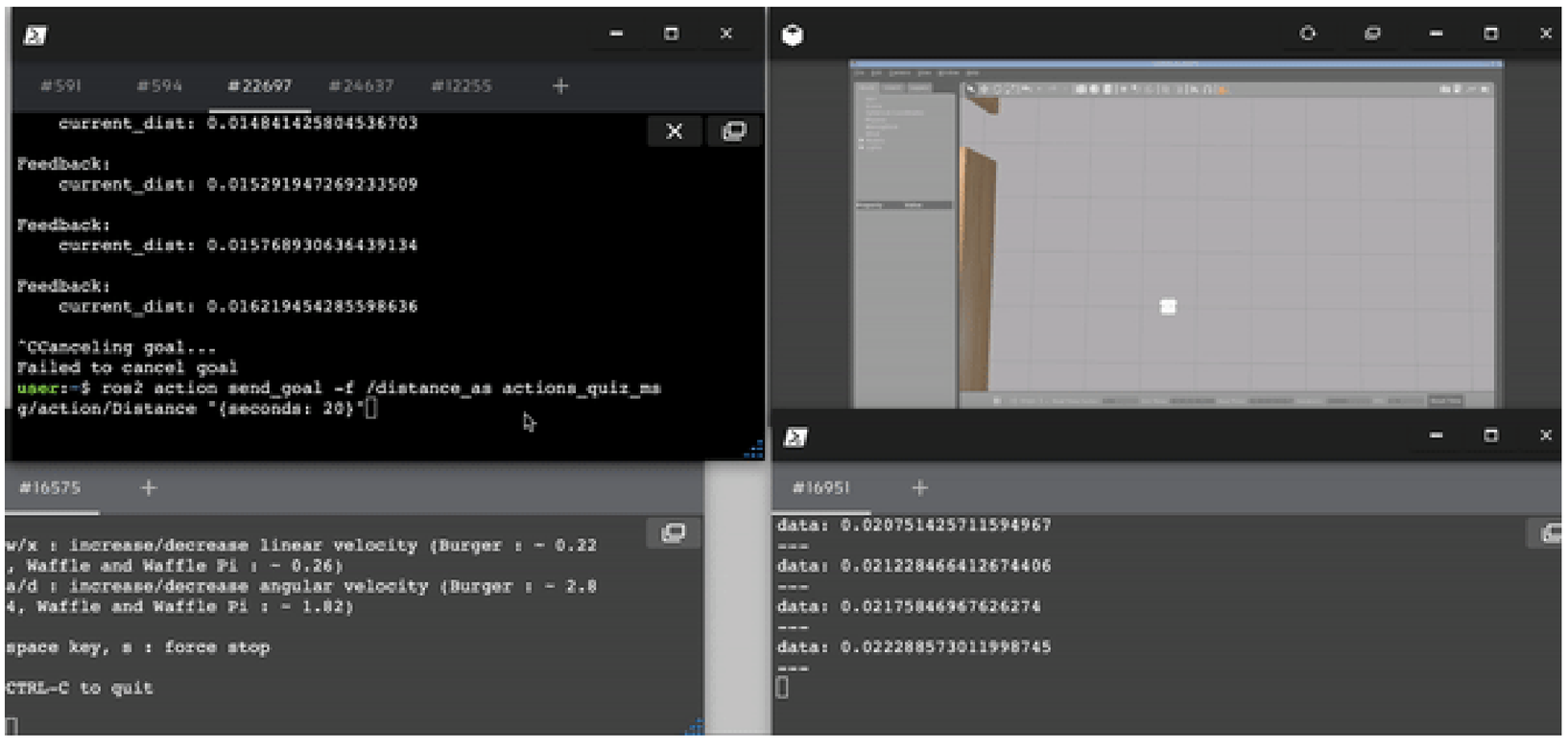
In [ ]:
```
from setuptools import setup
import os
from glob import glob

package_name = 'actions_quiz'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        (os.path.join('share', package_name), glob('launch/*.launch.py'))
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='user',
    maintainer_email='user@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'actions_quiz_server = actions_quiz.server:main',
            'actions_quiz_client = actions_quiz.client:main'
        ],
    },
)
```

In [ ]:
```
ros2 launch actions_quiz actions_quiz_server.launch.py
```

In [ ]:
```
ros2 action send_goal -f /distance_as actions_quiz_msg/action/Distance "{seconds: 20}"
```

Then, moving the robot using the `keyboard teleop` , you can observe:

- The current distance traveled at the **`/total_distance`** topic
- The current distance traveled as feedback in the client.
- When the time finishes, the topic publication stops and you get the final distance traveled as result



For reference, the Shell output looks something like this: