default

August 16, 2024

```
<hr />
<h1 class="text-center">
    <span class="text-primary">Checkpoint 10</span>
</h1>
<hr />
```

`- Summary -`

The purpose of this project is to learn how to program your ROS2 nodes into components, which is the recommended approach in ROS2.

For this, you will convert the code you created in **checkpoint9** into components and experience the differences, advantages and disadvantages between the 2 programming methods.

For this project you will use a simulation of the warehouse in which the RB1 robot will work. To launch it run the following commands:

Execute in Terminal

[ ]:
```
source ~/sim_ws/install/setup.bash
ros2 launch the_construct_office_gazebo warehouse_rb1.launch.xml
```

`- End of Summary -`

`- Prepare the environment -`

As already mentioned, in order to proceed with this Checkpoint you must have correctly finished Checkpoint 9. So, the first thing you have to do is to download the repository created for Checkpoint 9. Download your repository inside your **`ros2_ws`**.

Execute in Terminal

[ ]:
```
cd ~/ros2_ws/src
git clone <your_repository_name>
```

`- Prepare the environment -`

```
<h1 class="text-center">
    <span class="text-primary">Task 1</span>
    &nbsp;
    <span class="">Compose your nodes</span>
</h1>
```

For this first task, you will adapt the nodes you created for checkpoint 9 in order to work as components.

For achieving this, follow the next steps:

1. Inside the repository **`checkpoint9`**, create a new branch named **`composition`**.
2. In this new branch, add a new ROS2 package named **`my_components`**.
3. Create a new component named **`PreApproach`** that replicates the behavior of the **`pre_approach.cpp`** node you created in Checkpoint 9. This is, it moves the robot in front of the shelf, facing it (see image below).
4. For this component, you can remove the use of the parameters **`obstacle`** and **`degrees`** and use hardcoded values.

`- Grading Guide -`

- First of all, make sure you are in the `composition` branch:

Execute in Terminal

[ ]:
```
cd ~/ros2_ws/src/checkpoint9
git checkout composition
```

After you verify you are in the correct branch, build the workspace:

Execute in Terminal

[ ]:
```
cd ~/ros2_ws
colcon build
source install/setup.bash
```

- Start the container with the following command:

Execute in Terminal

[ ]:
```
ros2 run rclcpp_components component_container
```

- When you load the `PreApproach` container, the robot ends up right in front of the shelf - **3 points**

Execute in Terminal

[ ]:
```
ros2 component load /ComponentManager my_components my_components::PreApproach
```

`- End Grading Guide -`

```
<h1 class="text-center">
    <span class="text-primary">Task 2</span>
    &nbsp;
    <span class="">Final Approach</span>
</h1>
```

In this 2nd Task of the project, you will create 2 new components, a server and a client, that will allow the robot to attach to the shelf.

1. Create a new component named **`AttachServer`**. This component will contain the service server **`/approach_shelf`** that you created for Checkpoint 9. So, when called, the server will start the final approach behavior to make the robot attach to the shelf.
2. For the **`AttachServer`** component, use the **manual-composition** approach.
3. Create a new component named **`AttachClient`**. This component will contain a service client that will call to the `/approach_shelf` service so that the robot starts the final approach.
4. For the **`AttachClient`** component, use the **run-time composition** approach.
5. The service will use the same interface **`GoToLoading.srv`** that you created for Checkpoint 9.
6. For these components, you can remove the use of the **`final_approach`** parameter.
7. Create a launch file named **`attach_to_shelf.launch.py`**. This launch file will start a container named **`my_container`** and will load the following 2 components:
   - **`PreApproach`** with name **`pre_approach`**
   - **`AttachServer`** with name **`attach_server`**

`- Grading Guide -`

- Running the launch file **`attach_to_shelf.launch.py`** will make the robot move in front of the shelf, without attaching to it. - **1 point**

Execute in Terminal

```
[ ]: ros2 launch my_components attach_to_shelf.launch.py
```

- Verify the `/approach_shelf` service has also been started - **1 point**

Execute in Terminal

```
[ ]: ros2 service list | grep approach_shelf
```

Expected Output

- When loading the **`AttachClient`** component, the robot will attach to the shelf

Execute in Terminal

```
[ ]: ros2 component load /my_container my_components my_components::AttachClient
```

- Verify the robot correctly does the final approach and attaches to the shelf - **2 points**

`- End Grading Guide -`

```
<h1 class="text-center">
    <span class="text-primary">Extra</span>
    &nbsp;
    <span class="">Useful commands</span>
</h1>
```

You can send velocity commands to the mobile robot you are using by launching the following ROS2 node.

Execute in Terminal

[ ]:
```
ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args --remap cmd_vel:
↪=/diffbot_base_controller/cmd_vel_unstamped
```

REMEMBER, you need to focus on the terminal where you launched the program for the keys to take effect. You know you have correctly focused when the cursor starts blinking.

In order to lift the shelf you can use the following commands:

Lift the shelf:

Execute in Terminal

[ ]:
```
ros2 topic pub /elevator_up std_msgs/msg/String
```

Release the shelf:

Execute in Terminal

[ ]:
```
ros2 topic pub /elevator_down std_msgs/msg/String
```