Action client tutorial

Goal

The goal of this tutorial is to learn how to create an action client in ROS 2 in C++. This action client will send a goal to the action server */play_motion2* to start a predefined motion on TIAGo. Furthermore, a new predefined motion is created that can be launched through this action client.

ROS 2 tutorial: link

Clone the tutorial

Navigate the source folder of your ROS 2 workspace and clone the following package to get started:

```
git clone
https://agimus-user:frpTR_--SSsbKWRJkK5V@gitlab.com/pal-robotics/agimus_winte
r_school/tutorials/agimus_action_client_tutorial.git
```

Create a class

1.

First, in the header file, action_client.hpp, add the necessary dependencies, given below:

```
#include "rclcpp/rclcpp.hpp"
#include "rclcpp_action/rclcpp_action.hpp"
#include "play_motion2_msgs/action/play_motion2.hpp"
```

2.

Create a class, *PlayMotion2Client*, that inherits from the *rclcpp::Node* class. Use both the files *action_client.cpp* and *action_client.hpp* file. Create a simple constructor for this class initialised in the header and defined in the source file.

3.

To avoid lengthy lines add the following to the *PlayMotion2Client* class:

```
using PlayMotion2Action = play_motion2_msgs::action::PlayMotion2;
using GoalHandlePlayMotion2 = rclcpp_action::ClientGoalHandle<PlayMotion2Action>;
```

4.
Register the class as a component node as done in the ROS 2 tutorial. The advantage of using component nodes is that the node does not require a main function to be started.

5.
Add the following to CmakeLists.txt following the public ROS 2 tutorial. Instead of *ament_cmake*, use *ament_cmake_auto*, this simplifies the structure of CmakeLists.txt.

- Create a library that contains *action_client.cpp*.
- Register the node as a component node in the previously created library.

6.
Add the required dependencies to the *package.xml*:
- rclcpp
- rclcpp_actions
- rclcpp_components
- play_motion2_msgs

Create functions

1.
Following the structure of the public ROS 2 tutorial, create the following three functions. For this tutorial no feedback callback is required.

```
void send_goal();
void goal_response_callback(const GoalHandlePlayMotion2::SharedPtr & goal_handle);
void result_callback(const GoalHandlePlayMotion2::WrappedResult & result);
```

2.
To get more information about the action interface run the following command in a terminal, this shows the required input for the action goal and the received result:

```
ros2 interface show play_motion2_msgs/action/PlayMotion2
```

3.
In the constructor of *PlayMotion2Client*, declare a ROS parameter, called *motion_name*, and get the value of this parameter. Store the motion name as a member variable of the class.

4.
Ensure that the goal that is sent to the action client passes on the motion name of the member variable created in the previous step.

5.
Ensure the action client only sends the goal once.

Create launch file

Create a launch file, named *play_motion_client.launch.py*, that launches the previously created client node. This launch file does the following:

1. Declare a launch argument, *motion_name*.
2. Create the node that runs the action client from this tutorial. Add as a parameter the value of the launch argument.
3. Add both the launch argument and the node to the launch description.

To test the launch file, start a simulation of tiago. In another terminal, run the following:

```
ros2 launch agimus_action_client_tutorial play_motion_client.launch.py
motion_name:=home
```

Create a new motion

1.
Launch the simulation of tiago, with MoveIt! enabled:

```
ros2 launch tiago_gazebo tiago_gazebo.launch.py moveit:=true
```

2.
Open MoveIt! with RViz

```
ros2 launch tiago_moveit_config moveit_rviz.launch.py
```

Move the arm of TIAGo using the MoveIt plugin into the desired states (key frames). For every desired key frame, save the joint states using the subscriber from the previous tutorial.

3.
Add the saved joint states in the file:

```
sudo vim
/opt/pal/alum/share/tiago_bringup/config/motions/tiago_motions_pal-gripper_schunk-ft.
yaml
```

Use the layout given below to. Note that the *positions* should be a matrix of size K x J, where K is the number of key frames and J is the number of joints. The vector *time_from_start* should have a length of K.

```
custom_motion_name:
   joints: [arm_1_joint,
   arm_2_joint, arm_3_joint, arm_4_joint, arm_5_joint,
   arm_6_joint, arm_7_joint]
   positions: [0.20, 0.35, -0.20, 1.94, -1.57, 1.37, 0.0,
               0.20, -1.34, -0.20, 1.94, -1.57, 1.37, 0.0,
               0.20, -1.34, -0.20, 1.94, -1.57, 1.37, 0.0]
   times_from_start: [0.5, 4.0, 7.0]
   meta:
     name: Custom
     usage: demo
     description: 'This is your custom motion'
```

4.

Once a motion has been created and saved, restart the simulation of tiago and launch the action client of this tutorial, with the custom motion name as launch argument.

```
ros2 launch agimus_action_client_tutorial play_motion_client.launch.py
motion_name:=custom_motion_name
```