Example #1: Simple Action Server

Aim

- To write a ROS Node which will act as Simple Action Server. This Simple Action Server should be able to process goals coming from Simple Action Client.
- The role of this server should be to move and rotate the turtle in `turtlesim_node` of `turtlesim` package.
- The server should accept **goal** which should have **distance** by which the turtle should be moved and the **angle** by which it should be rotated to face the direction of motion. For eg. if the goal is distance=1 and angle=90 then the turtle should rotate by 90 degree then move by a distance of 1 unit.
- The server should give current pose of the turtle (**x, y, theta**) as **feedback** to the client.
- The server should give final pose of the turtle (**x, y, theta**) as **result** to the client.
- Create this Simple Action Server Node in `pkg_ros_actions` ROS Package inside `scripts` folder and then make is executable using `chmod` command.
- The name of the action use by this Simple Action Server should be `/action_turtle`.

NOTE: Same action file discussed at **Create a action message file** section can be used here also.

Code

`node_simple_action_server_turtle.py`

```
#!/usr/bin/env python

# ROS Node - Simple Action Server - Turtle

import rospy
import actionlib
import math
import time

from turtlesim.msg import Pose
from geometry_msgs.msg import Twist

from pkg_ros_actions.msg import myActionMsgAction       # Message Class that is used by F
from pkg_ros_actions.msg import myActionMsgGoal         # Message Class that is used for
from pkg_ros_actions.msg import myActionMsgResult       # Message Class that is used for
from pkg_ros_actions.msg import myActionMsgFeedback     # Message Class that is used for


class SimpleActionServerTurtle:

    # Constructor
    def __init__(self):

        # Initialize Simple Action Server
        self._sas = actionlib.SimpleActionServer('/action_turtle',
                                                  myActionMsgAction,
                                                  execute_cb=self.func_on_rx_goal,
                                                  auto_start=False)
        '''
        * '/action_turtle' - The name of the action that will be used by ROS Nodes to con
        * myActionMsgAction - The Message Class that is used by ROS Actions internally fo
        * execute_cb - Holds the function pointer to the function which will process inco
        * auto_start = False - Only when self._sas.start() will be called then only this
        '''

        # Declare constants
        self._config_ros_pub_topic = '/turtle1/cmd_vel'
        self._config_ros_sub_topic = '/turtle1/pose'

        # Declare variables
        self._curr_x = 0
        self._curr_y = 0
        self._curr_theta = 0

        # Start the Action Server
        self._sas.start()
        rospy.loginfo("Started Turtle Simple Action Server.")

    #-------------------------------------------------------
    # Callback Function for ROS Topic ('/turtle1/pose') Subscription
    def func_ros_sub_callback(self, pose_message):
        self._curr_x = pose_message.x
        self._curr_y = pose_message.y
        self._curr_theta = pose_message.theta

    #-------------------------------------------------------
    # Function to move the turtle in turtlesim_node straight
    def func_move_straight(self, param_dis, param_speed, param_dir):

        obj_velocity_mssg = Twist()
        obj_pose_mssg = Pose()

        # Store the start position of the turtle
        start_x = self._curr_x
        start_y = self._curr_y

        # Move the turtle till it reaches the desired position by publishing to Velocity
        handle_pub_vel = rospy.Publisher(
            self._config_ros_pub_topic, Twist, queue_size=10)

        # 1 Hz : Loop will its best to run 1 time in 1 second
        var_loop_rate = rospy.Rate(10)

        # Set the Speed of the Turtle according to the direction
        if(param_dir == 'b'):
            obj_velocity_mssg.linear.x = (-1) * abs(int(param_speed))
        else:
            obj_velocity_mssg.linear.x = abs(int(param_speed))

        # Move till desired distance is covered
        dis_moved = 0.0

        while not rospy.is_shutdown():

            # Send feedback to the client
            obj_msg_feedback = myActionMsgFeedback()

            obj_msg_feedback.cur_x = self._curr_x
            obj_msg_feedback.cur_y = self._curr_y
            obj_msg_feedback.cur_theta = self._curr_theta

            self._sas.publish_feedback(obj_msg_feedback)

            if ((dis_moved < param_dis)):
                handle_pub_vel.publish(obj_velocity_mssg)

                var_loop_rate.sleep()

                dis_moved = abs(
                    math.sqrt(((self._curr_x - start_x) ** 2) + ((self._curr_y - start_y)
                print('Distance Moved: {}'.format(dis_moved))
            else:
                break

        # Stop the Turtle after desired distance is covered
        obj_velocity_mssg.linear.x = 0
        handle_pub_vel.publish(obj_velocity_mssg)
        print('Destination Reached')

    #-------------------------------------------------------
    # Function to rotate the turtle in turtlesim_node
    def func_rotate(self, param_degree, param_speed, param_dir):

        obj_velocity_mssg = Twist()
        obj_pose_mssg = Pose()

        # Store start Theta of the turtle
        start_degree = abs(math.degrees(self._curr_theta))
        current_degree = abs(math.degrees(self._curr_theta))

        # Rotate the turtle till desired angle is reached
        handle_pub_vel = rospy.Publisher(
            self._config_ros_pub_topic, Twist, queue_size=10)

        # 1 Hz : Loop will its best to run 1 time in 1 second
        var_loop_rate = rospy.Rate(10)

        # Set the speed of rotation according to param_dir
        if(param_dir == 'a'):
            obj_velocity_mssg.angular.z = math.radians(
                abs(int(param_speed)))  # Anticlockwise
        else:
            # Clockwise
            obj_velocity_mssg.angular.z = (-1) * \
                math.radians(abs(int(param_speed)))

        # Rotate till desired angle is reached
        degree_rotated = 0.0

        while not rospy.is_shutdown():
            if((round(degree_rotated) < param_degree)):
                handle_pub_vel.publish(obj_velocity_mssg)

                var_loop_rate.sleep()

                current_degree = abs(math.degrees(self._curr_theta))
                degree_rotated = abs(current_degree - start_degree)
                print('Degree Rotated: {}'.format(degree_rotated))
            else:
                break

        # Stop the Turtle after the desired angle is reached
        obj_velocity_mssg.angular.z = 0
        handle_pub_vel.publish(obj_velocity_mssg)
        print('Angle Reached')

    #-------------------------------------------------------
    # Function to process Goals and send Results
    def func_on_rx_goal(self, obj_msg_goal):
        rospy.loginfo("Received a Goal from Client.")
        rospy.loginfo(obj_msg_goal)

        flag_success = False        # Set to True if Goal is successfully achieved
        flag_preempted = False      # Set to True if Cancel req is sent by Client

        # --- Goal Processing Section ---
        self.func_rotate(obj_msg_goal.angle, '10', 'a')
        self.func_move_straight(obj_msg_goal.distance, '1', 'f')

        # Send Result to the Client
        obj_msg_result = myActionMsgResult()
        obj_msg_result.final_x = self._curr_x
        obj_msg_result.final_y = self._curr_y
        obj_msg_result.final_theta = self._curr_theta

        rospy.loginfo("send goal result to client")
        self._sas.set_succeeded(obj_msg_result)


# Main Function
def main():
    # 1. Initialize ROS Node
    rospy.init_node('node_simple_action_server_turtle')

    # 2. Create Simple Action Server object.
    obj_server = SimpleActionServerTurtle()

    # 3. Subscribe to Pose of the Turtle
    handle_sub_pose = rospy.Subscriber(obj_server._config_ros_sub_topic, Pose, obj_server

    # 4. Do not exit and loop forever.
    rospy.spin()


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

Download

Run Command

Now this server do the following,

```
roscd pkg_ros_actions

cd srcipts

sudo chmod +x node_simple_action_server_turtle.py

rosrun pkg_ros_actions node_simple_action_server_turtle.py
```

NOTE: `roscore` should be running if you want to run a ROS Node.