Text Large Model + Robotic Arm Control（Text Version)

Before running the function, you need to close the App and large programs. For the closing method, refer to [4. Preparation] - [1. Manage APP control services].

1. Function Description

After the program runs, you can input a series of action commands through the terminal. The large model will plan the corresponding action functions for these commands and execute all instructions sequentially.

2. Startup

Users with Jetson-Nano board version need to enter the docker container and input the following command. Orin board users can directly open the terminal and input the following command:

```
ros2 launch largemodel largemodel_control.launch.py text_chat_mode:=True
```

Then open a second terminal and input the following command:

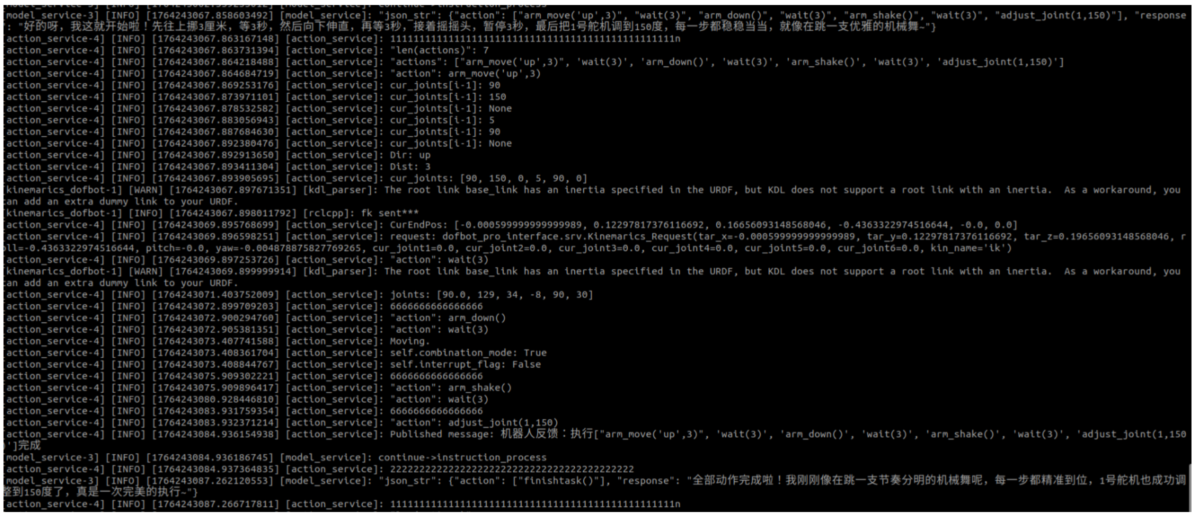
```
ros2 run text_chat text_chat
```

Then input the commands you want to control the robotic arm in the text_chat terminal. Currently, the following robotic arm control commands are available:

- Individual servo control: Set servo x to y degrees, where x ranges from 1-6 and y ranges from 0-180, for example, set servo 1 to 120 degrees.
- Individual robotic arm action control: Robotic arm up/down/left/right, gripping posture, tracking posture, for example, set the robotic arm to gripping posture, robotic arm extend upward;
- Control robotic arm motion action groups: Robotic arm dance/clap/nod/shake head, for example, please nod your head;
- Change robotic arm end position: Robotic arm adjust up/down/left/right/forward/backward by x centimeters, for example, adjust the robotic arm upward by 3 centimeters.

You can freely combine robotic arm control commands, and the large model will plan the corresponding action groups for each command. You can refer to the following content and input in the text_chat terminal:

```
First, adjust the robotic arm upward by 3 centimeters, wait 3 seconds, then
extend the robotic arm downward, stay for 3 seconds, then have the robotic arm
shake its head, pause for 3 seconds, and finally set servo 1 to 150 degrees.
```



```
model_service-3] [INFO] [1764243067.858603492] [model_service]: "json_str": {"action": ["arm_move('up',3)", "wait(3)", "arm_down()", "wait(3)", "arm_shake()", "wait(3)", "adjust_joint(1,150)"], "response
": "好的呀，我这就开始啦！先往上挪3厘米，等3秒，然后向下伸直，再等3秒，接着摇摇头，暂停3秒，最后把1号舵机调到150度，每一步都稳稳当当，就像在跳一支优雅的机械舞~"}
action_service-4] [INFO] [1764243067.863167148] [action_service]: 1111111111111111111111111111111111111111111111111n
action_service-4] [INFO] [1764243067.863731394] [action_service]: "len(actions)": 7
action_service-4] [INFO] [1764243067.864218488] [action_service]: "actions": ["arm_move('up',3)", 'wait(3)', 'arm_down()', 'wait(3)', 'arm_shake()', 'wait(3)', 'adjust_joint(1,150)']
action_service-4] [INFO] [1764243067.864684719] [action_service]: "action": arm_move('up',3)
action_service-4] [INFO] [1764243067.869253176] [action_service]: cur_joints[i-1]: 90
action_service-4] [INFO] [1764243067.873971101] [action_service]: cur_joints[i-1]: 150
action_service-4] [INFO] [1764243067.878532582] [action_service]: cur_joints[i-1]: None
action_service-4] [INFO] [1764243067.883056943] [action_service]: cur_joints[i-1]: 5
action_service-4] [INFO] [1764243067.887684630] [action_service]: cur_joints[i-1]: 90
action_service-4] [INFO] [1764243067.892380476] [action_service]: cur_joints[i-1]: None
action_service-4] [INFO] [1764243067.892913650] [action_service]: Dir: up
action_service-4] [INFO] [1764243067.893411304] [action_service]: Dist: 3
action_service-4] [INFO] [1764243067.893905695] [action_service]: cur_joints: [90, 150, 0, 5, 90, 0]
kinemarics_dofbot-1] [WARN] [1764243067.897671351] [kdl_parser]: The root link base_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia.  As a workaround, you
can add an extra dummy link to your URDF.
kinemarics_dofbot-1] [INFO] [1764243067.898011792] [rclcpp]: fk sent***
action_service-4] [INFO] [1764243069.895768699] [action_service]: CurEndPos: [-0.000599999999999989, 0.12297817376116692, 0.16656093148568046, -0.4363322974516644, -0.0, 0.0]
action_service-4] [INFO] [1764243069.896598251] [action_service]: request: dofbot_pro_interface.srv.Kinematics_Request(tar_x=-0.000599999999999989, tar_y=0.12297817376116692, tar_z=0.19656093148568046, r
oll=-0.4363322974516644, pitch=-0.0, yaw=-0.004878875827769265, cur_joint1=0.0, cur_joint2=0.0, cur_joint3=0.0, cur_joint4=0.0, cur_joint5=0.0, cur_joint6=0.0, kin_name='ik')
action_service-4] [INFO] [1764243069.897253726] [action_service]: "action": wait(3)
kinemarics_dofbot-1] [WARN] [1764243069.899999914] [kdl_parser]: The root link base_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia.  As a workaround, you
can add an extra dummy link to your URDF.
action_service-4] [INFO] [1764243071.403752009] [action_service]: joints: [90.0, 129, 34, -8, 90, 30]
action_service-4] [INFO] [1764243072.899709203] [action_service]: 6666666666666666
action_service-4] [INFO] [1764243072.900294760] [action_service]: "action": arm_down()
action_service-4] [INFO] [1764243072.905381351] [action_service]: "action": wait(3)
action_service-4] [INFO] [1764243073.407741588] [action_service]: Moving.
action_service-4] [INFO] [1764243073.408361704] [action_service]: self.combination_mode: True
action_service-4] [INFO] [1764243073.408844767] [action_service]: self.interrupt_flag: False
action_service-4] [INFO] [1764243075.909302221] [action_service]: 6666666666666666
action_service-4] [INFO] [1764243075.909896417] [action_service]: "action": arm_shake()
action_service-4] [INFO] [1764243080.928446810] [action_service]: "action": wait(3)
action_service-4] [INFO] [1764243083.931759354] [action_service]: 6666666666666666
action_service-4] [INFO] [1764243083.932371214] [action_service]: "action": adjust_joint(1,150)
action_service-4] [INFO] [1764243084.936154938] [action_service]: Published message: 机器人反馈：执行["arm_move('up',3)", 'wait(3)', 'arm_down()', 'wait(3)', 'arm_shake()', 'wait(3)', 'adjust_joint(1,150
)']完成
model_service-3] [INFO] [1764243084.936186745] [model_service]: continue->instruction_process
action_service-4] [INFO] [1764243084.937364835] [action_service]: 2222222222222222222222222222222222222222222
model_service-3] [INFO] [1764243087.262120553] [model_service]: "json_str": {"action": ["finishtask()"], "response": "全部动作完成啦！我刚刚像在跳一支节奏分明的机械舞呢，每一步都精准到位，1号舵机也成功调
整到150度了，真是一次完美的执行~"}
action_service-4] [INFO] [1764243087.266717811] [action_service]: 1111111111111111111111111111111111111111111111111n
```

3. Core Code Analysis

Mainly look at the servo angle setting adjust_joint function. Other functions were analyzed in the source code analysis of the first lesson [AI Model - Text Version\1. Semantic Understand Instruction Following] and will not be repeated here.

The adjust_joint function source code path is: `LargeModel_ws/src/largemodel/largemodel/action_service.py`

```python
#Pass two parameters, joint_id is the servo number, ranging from 1-6, angle is the angle value, ranging from 0-180
def adjust_joint(self,joint_id,angle):
    #Directly communicate with the top-level driver board, send the servo value and angle value to be set
    Arm.Arm_serial_servo_write(int(joint_id), int(angle), 1000)
    time.sleep(1.0)
    if not self.combination_mode and not self.interrupt_flag:
        self.action_status_pub("adjust_joint_done")
```