Semantic Understanding and Instruction Following（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, a series of action commands are entered through the terminal. The large language model will plan the corresponding action functions for these commands and execute all instructions sequentially.

2. Startup

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

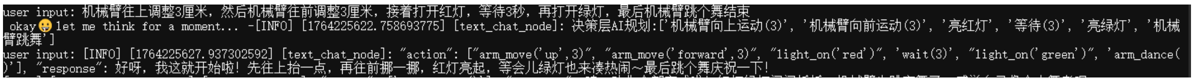
```
ros2 launch largemodel largemodel_control.launch.py text_chat_mode:=True
```

Then open a second terminal and enter the following command:

```
ros2 run text_chat text_chat
```

Then, in the text_chat terminal, enter the desired action commands. Refer to the following example:

Move the robotic arm up 3 centimeters, then move the robotic arm forward 3 centimeters, then turn on the red light, wait 3 seconds, then turn on the green light, and finally have the robotic arm perform a dance.



```
user input: 机械臂往上调整3厘米，然后机械臂往前调整3厘米，接着打开红灯，等待3秒，再打开绿灯，最后机械臂跳个舞结束
 okay😛let me think for a moment... -[INFO] [1764225622.758693775] [text_chat_node]: 决策层AI规划:['机械臂向上运动(3)', '机械臂向前运动(3)', '亮红灯', '等待(3)', '亮绿灯', '机械
臂跳舞']
user input: [INFO] [1764225627.937302592] [text_chat_node]: "action": ["arm_move('up',3)", "arm_move('forward',3)", "light_on('red')", 'wait(3)', "light_on('green')", 'arm_dance(
)'], "response": 好呀，我这就开始啦！先往上抬一点，再往前挪一挪，红灯亮起，等会儿绿灯也来凑热闹～最后跳个舞庆祝一下！
```

As shown in the image above, the large language model will plan a series of action command functions. The corresponding action commands are: **"arm_move('up',3)", "arm_move('forward',3)", "light_on('red')", 'wait(3)', "light_on('green')", 'arm_dance()'**.

The robotic arm end effector will first move forward 3 centimeters, then the LED on the driver board will light up red, wait three seconds, the LED on the driver board will light up green, and finally the robotic arm will perform a dance.

3. Core Code Analysis

3.1. arm_move function

`Source code path: LargeModel_ws/src/largemodel/largemodel/action_service.py`

```
#Dir indicates the direction that needs adjustment, and Dist indicates the distance that needs adjustment.
def arm_move(self,Dir,Dist):
    self.arm_move_flag = True
    cur_joints = [0.0,0.0,0.0,0.0,0.0,0.0]
    #Get the current angle values of the six servos
    for i in range(1,7):
        cur_joints[i-1] = Arm.Arm_serial_servo_read(i)
        self.get_logger().info(f"cur_joints[i-1]: {cur_joints[i-1]}")
        if cur_joints[i-1] == None:
            cur_joints[i-1] = 0
            #self.get_logger().info('Servo Reading...')
    self.cur_joints = cur_joints
    Dir = Dir.strip("'\"")  # Remove single and double quotes
    self.Dir = Dir
    Dist = int(Dist)
    self.get_logger().info(f"Dir: {Dir}")
    self.get_logger().info(f"Dist: {Dist}")
    self.get_logger().info(f"cur_joints: {cur_joints}")
    self.cur_joints[0] = float(self.cur_joints[0])
    self.cur_joints[1] = float(self.cur_joints[1])
    self.cur_joints[2] = float(self.cur_joints[2])
    self.cur_joints[3] = float(self.cur_joints[3])
    self.cur_joints[4] = float(self.cur_joints[4])
    self.cur_joints[5] = float(self.cur_joints[5])
    #Get the end-effector pose of the current robotic arm
    self.get_current_end_pos()
    time.sleep(2)
    self.get_logger().info(f"CurEndPos: {self.CurEndPos}")
 #Call the move function to perform calculations and adjustments.
    self.move(Dir,Dist)

def move(self,Dir,Dist):
    while not self.client.wait_for_service(timeout_sec=1.0):
        self.get_logger().info('Service not available, waiting again...')

    request = Kinemarics.Request()
  #Determine the direction and calculate the target end position of the robotic
arm based on the direction and the required adjustment distance.
    if Dir == 'left':
        request.tar_x = self.CurEndPos[0] - Dist*0.01
    elif Dir == 'right':
        request.tar_x = self.CurEndPos[0] + Dist*0.01
    else:
        request.tar_x = self.CurEndPos[0]
    if Dir == 'forward':
        request.tar_y = self.CurEndPos[1]  + Dist*0.01
    elif Dir == 'backwards':
        request.tar_y = self.CurEndPos[1]  - Dist*0.01
    else:
        request.tar_y = self.CurEndPos[1]
    if Dir == 'up':
        request.tar_z = self.CurEndPos[2] + Dist*0.01
    elif Dir == 'down':
        request.tar_z = self.CurEndPos[2] - Dist*0.01
    else:
        request.tar_z = self.CurEndPos[2]
    request.kin_name = "ik"
    request.roll = self.CurEndPos[3]
    request.pitch = self.CurEndPos[4]
    request.yaw = math.atan(request.tar_x/request.tar_y)
    self.get_logger().info(f"request: {request}")
    future = self.client.call_async(request)

#Call the inverse kinematics service to calculate the target pose of the robotic
arm.
 def get_ik_respone_callback(self, future):
    try:
        response = future.result()
        joints = [0.0, 0.0, 0.0, 0.0, 0.0,0.0]
        joints[0] = int(response.joint1) #response.joint1
        joints[1] = int(response.joint2)
        joints[2] = int(response.joint3)
        if response.joint4>90:
            joints[3] = 90
        else:
            joints[3] = int(response.joint4)
        joints[4] = 90
        joints[5] = 30
        time.sleep(1.5)
        #If adjusting left or right, only change the value of servo motor number
one; keep all other values unchanged.
        if self.Dir == 'left' or self.Dir == 'right':
            joints[1] = self.cur_joints[1]
            joints[2] = self.cur_joints[2]
            joints[3] = self.cur_joints[3]
         #If adjusting up, down, forward, or backward, keep the value of servo
motor number one constant, and change the other values.
        elif  self.Dir == 'down' or self.Dir == 'up' or self.Dir == 'forward' or
self.Dir == 'backwards':
            joints[0] = self.cur_joints[0]
        self.get_logger().info(f"joints: {joints}")
        if self.return_flag == True:
            Arm.Arm_serial_servo_write6(joints[0], joints[1], joints[2],
joints[3], 90, self.grasp_joint,2000)
            time.sleep(2.0)
            Arm.Arm_serial_servo_write(6, 0, 2000)
            time.sleep(2.0)
        else:
            #Communicating with the lower-level control board to control the
robotic arm's movement
            Arm.Arm_serial_servo_write6(joints[0], joints[1], joints[2],
joints[3], 90, self.grasp_joint,2000)
            time.sleep(2.0)
            for i in range(6):
                if joints[i] <0:
                    joints[i] = 0
            self.cur_joints = joints
        if self.arm_move_flag == False:
            Arm.Arm_serial_servo_write6(90,120,10,10,90,30,2000)
            time.sleep(2.0)
            if self.back_list == False:
                self.action_status_pub("return_to_orin_done")
            self.return_done = True
        else:
            #time.sleep(2.0)
            self.get_logger().info("Moving.")
            self.get_logger().info(f"self.combination_mode:
{self.combination_mode}")
```

```
            self.get_logger().info(f"self.interrupt_flag:
{self.interrupt_flag}")
            if not self.combination_mode and not self.interrupt_flag:
                self.get_logger().info("Move done.")
                self.action_status_pub("arm_move_done")
    except Exception:
       pass
```

3.2. The `light_on` function

`Source code path: LargeModel_ws/src/largemodel/largemodel/action_service.py`

```
def light_on(self,color):
    color = color.strip("'\"")  # Remove single and double quotes
    self.get_logger().info("Trun on the RGB Light.")
    if color == "red":
        self.get_logger().info("Trun on the Red Light.")
        Arm.Arm_RGB_set(50, 0, 0) #RGB bright red light
    elif color == "green":
        self.get_logger().info("Trun on the Green Light.")
        Arm.Arm_RGB_set(0, 50, 0) #RGB bright green light
    elif color == "blue":
        self.get_logger().info("Trun on the Blue Light.")
        Arm.Arm_RGB_set(0, 0, 50) #RGB bright blue light
    if not self.combination_mode and not self.interrupt_flag:
        self.action_status_pub("light_on_done")
```

3.3. The `wait` function

`Source code path: LargeModel_ws/src/largemodel/largemodel/action_service.py`

```
def wait(self, duration):
    duration = float(duration)
   #Sleep for duration seconds
    time.sleep(duration)
    if not self.combination_mode and not self.interrupt_flag:
        self.action_status_pub("wait_done", duration=duration)
```

3.4. arm_dance function

`Source code path: LargeModel_ws/src/largemodel/largemodel/action_service.py`

```
# Robotic Arm Dancing: Communicating directly with the underlying control board
to control the robotic arm's movement to a specified posture.
def arm_dance(self):
    Arm.Arm_serial_servo_write6(90,90,90,90,90,90,1000)
    time.sleep(1.0)
    Arm.Arm_serial_servo_write6(90,60,120,60,90,90,1000)
    time.sleep(1.0)
    Arm.Arm_serial_servo_write6(90,45,135,45,90,90,1000)
    time.sleep(1.0)
    Arm.Arm_serial_servo_write6(90,60,120,60,90,90,1000)
    time.sleep(1.0)
    Arm.Arm_serial_servo_write6(90,90,90,90,90,90,1000)
    time.sleep(1.0)
    Arm.Arm_serial_servo_write6(90,100,80,80,90,90,1000)
    time.sleep(1.0)
    Arm.Arm_serial_servo_write6(90,120,60,60,90,90,1000)
    time.sleep(1.0)
    Arm.Arm_serial_servo_write6(90, 135, 45, 45, 90, 90,1000)
    time.sleep(1.0)
    Arm.Arm_serial_servo_write6(90,90,90,90,90,90,1000)
    time.sleep(1.0)
    Arm.Arm_serial_servo_write6(90, 90, 90, 20, 90, 150,1000)
    time.sleep(1.0)
    Arm.Arm_serial_servo_write6(90, 90, 90, 90, 90, 90,1000)
    time.sleep(1.0)
    Arm.Arm_serial_servo_write6(90, 90, 90, 20, 90, 150,1000)
    time.sleep(1.0)
    Arm.Arm_serial_servo_write6(90, 130, 0, 5, 90, 0,1000)
    if not self.combination_mode and not self.interrupt_flag:
        self.action_status_pub("arm_dance_done")
```