Multimodal Large Model + Robotic Arm Color Block Handling（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 color block handling commands through the terminal. The large model will plan the actions to complete the color block handling, then the program will control the robotic arm to handle the color blocks in sequence.

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
```

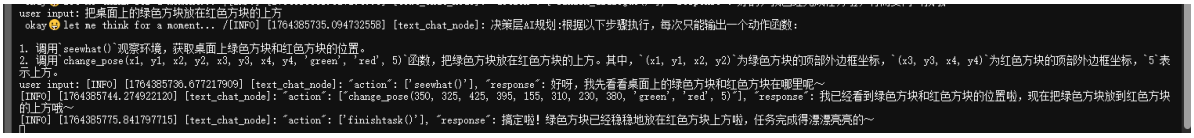
Then open a second terminal and input the following command:

```
ros2 run text_chat text_chat
```

Then input color block handling commands in the terminal. You can refer to the following example:

```
Place the green block on the desktop above the red block
```

The placement directions can be front, back, left, right, or top - five directions in total.



The program will first take a photo to get the positions of the green block and red block; then it will control the robotic arm to grip the green block and place it above the red block.

3. Task Planning

1. Call `seewhat()` to observe the environment and get the positions of the green block and red block on the desktop.
2. Call the `change_pose(x1, y1, x2, y2, x3, y3, x4, y4, 'green', 'red', 5)` function to place the green block above the red block. Where `(x1, y1, x2, y2)` are the top outer border coordinates of the green block, `(x3, y3, x4, y4)` are the top outer border coordinates of the red block, and `5` means above.

4. Core Code Analysis

4.1. change_pose Function

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

```
#Parameter description
#x1,y1,x2,y2: Top-left and bottom-right coordinates of the outer border of the color block to be gripped, in the above example it's the outer border coordinates of the green block
#x3,y3,x4,y4: Top-left and bottom-right coordinates of the outer border of the color block to be placed, in the above example it's the outer border coordinates of the red block
#src: Color of the block to be gripped, in the above example it's the green block
#tar: Color of the block to be placed, in the above example it's the red block
#side: Placement direction, front, back, left, right, top correspond to values 1, 2, 3, 4, 5 respectively, in the above example it's 5, meaning top
def change_pose(self, x1, y1, x2, y2,x3,y3,x4,y4,src,tar,side):
    #Extract the color of the block to be gripped
    src_color = src.strip("'\"")
    self.cur_down_name = src_color
    #Create a key in the dictionary
    self.cur_down_pose[self.cur_down_name] = []
    #Start the handling program
    cmd1 = "ros2 run largemodel_arm Get_Target_Pose_KCF"
    subprocess.Popen(
        [
            "gnome-terminal",
            "--title=grasp_desktop",
            "--",
            "bash",
            "-c",
            f"{cmd1}; exec bash",
        ]
    )
    time.sleep(3.0)
    #Publish outer border coordinate information and placement direction topic data, Get_Target_Pose_KCF node will subscribe to this topic
    x1 = int(x1)
    y1 = int(y1)
    x2 = int(x2)
    y2 = int(y2)
    x3 = int(x3)
    y3 = int(y3)
    x4 = int(x4)
    y4 = int(y4)
    side = int(side)
    time.sleep(5.0)
    self.object_position_pub.publish(Int16MultiArray(data=[x1, y1, x2, y2,x3,y3,x4,y4,side])
```

4.2. Get_Target_Pose_KCF Handling Program

Source code path:
`LargeModel_ws/src/largemodel_arm/largemodel_arm/Get_Target_Pose_KCF.py`

```
#Create subscriber, subscribe to corner_xy topic content
self.xy_subscription =
self.create_subscription(Int16MultiArray,'corner_xy',self.GetXYCallback,qos_prof
ile=1)
#Create publisher, publish topic of robotic arm placement posture angle values
self.down_joints_pub = self.create_publisher(CurJoints,'/down_joints',1)
#GetXYCallback callback function, handles received topic message data
def GetXYCallback(self,msg):
    print("msg: ",msg.data)
    #Get outer border coordinates of the block to be gripped
    self.Roi_init_src = (msg.data[0],msg.data[1],msg.data[2],msg.data[3])
    #Get outer border coordinates of the block to be placed
    self.Roi_init_tar = (msg.data[4],msg.data[5],msg.data[6],msg.data[7])
    #Get placement direction data
    self.side = msg.data[8]
    self.Track_state = 'identify'
    self.get_xy = True
    time.sleep(2.5)

#Image callback function get_Src_Tar
def get_Src_Tar(self,color_frame,depth_frame):
    #If outer border coordinates of both gripped and placed blocks are not empty
    if len(self.Roi_init_src)!=0 and len(self.Roi_init_tar)!=0:
        cv2.rectangle(result_image, (self.Roi_init_src[0],
self.Roi_init_src[1]), (self.Roi_init_src[2], self.Roi_init_src[3]), (255,0,0),
thickness=2)
        cv2.rectangle(result_image, (self.Roi_init_tar[0],
self.Roi_init_tar[1]), (self.Roi_init_tar[2], self.Roi_init_tar[3]), (0,255,0),
thickness=2)

        #Get center point coordinates of the gripped block and calculate its
depth information
        grasp_cx = (self.Roi_init_src[0] + self.Roi_init_src[2])/2
        grasp_cy = (self.Roi_init_src[1] + self.Roi_init_src[3])/2
        grasp_depth = depth_image_info[int(grasp_cy),int(grasp_cx)]/1000

        #Get center point coordinates of the placed block and calculate its
depth information
        tar_cx = (self.Roi_init_tar[0] + self.Roi_init_tar[2])/2
        tar_cy = (self.Roi_init_tar[1] + self.Roi_init_tar[3])/2
        tar_depth = depth_image_info[int(tar_cy),int(tar_cx)]/1000
        #If depth information of both is not 0, it means the distance is valid
        if tar_depth!=0 and grasp_depth!=0:
            #Calculate position information of the gripped block
            pose_grasp = self.compute_heigh(grasp_cx,grasp_cy,grasp_depth)
            #Calculate position information of the placed block
            pose_tar = self.compute_heigh(tar_cx,tar_cy,tar_depth)
            self.tar_x = pose_tar[0]
            self.tar_y = pose_tar[1]
            self.tar_z = pose_tar[2]
            #print("pose: ",pose)
            if self.done ==True:
```

```
                #Get current end effector position of the robotic arm
                self.get_current_end_pos()
                self.done = False
                #Call grip function to grip the block, passing in position
information of the gripped block
                self.grasp(pose_grasp[0],pose_grasp[1],pose_grasp[2])
#Grip function grasp, passing in position information of gripped block, calling
inverse kinematics service
def grasp(self,x,y,z):
    while not self.client.wait_for_service(timeout_sec=1.0):
        self.get_logger().info('Service not available, waiting again...')
    self.get_current_end_pos()
    request = Kinemarics.Request()
    request.tar_y = y  + self.y_offset + 0.01
    request.tar_z = z  + self.z_offset + 0.01
    if x>0:
        request.tar_x = x +0.01 + self.x_offset
    else:
        request.tar_x = x - 0.01 + self.x_offset
    request.kin_name = "ik"
    request.roll = -0.785
    future = self.client.call_async(request)
    future.add_done_callback(self.grasp_get_ik_respone_callback)
#Inverse kinematics service callback function grasp_get_ik_respone_callback, gets
servo values, controls servos to move to target posture and grip
def grasp_get_ik_respone_callback(self, future):
    try:
        response = future.result()
        print("calcutelate_response: ",response)
        joints = [0.0, 0.0, 0.0, 0.0, 0.0,0.0]
        joints[0] = response.joint1 #response.joint1
        joints[1] = response.joint2
        joints[2] = response.joint3
        if response.joint4>90:
            joints[3] = 90
        else:
            joints[3] = response.joint4
        joints[4] = 90
        joints[5] = 30

 self.Arm.Arm_serial_servo_write6(joints[0],joints[1],joints[2],joints[3],joints
[4],joints[5],2000)
        time.sleep(3.5)
        self.Arm.Arm_serial_servo_write(6, 135, 2000)
        time.sleep(2.5)
        self.init_joints[5] = self.grasp_joint
        #After gripping, return to initial posture, prepare to place above the
placed block
        self.Arm.Arm_serial_servo_write6_array(self.init_joints,2000)
        time.sleep(2.5)
        #Execute function to place above the placed block
        self.move()
    except Exception:
       pass
#Place block function move
def move(self):
    request = Kinemarics.Request()
    px = self.tar_x + self.x_offset
```

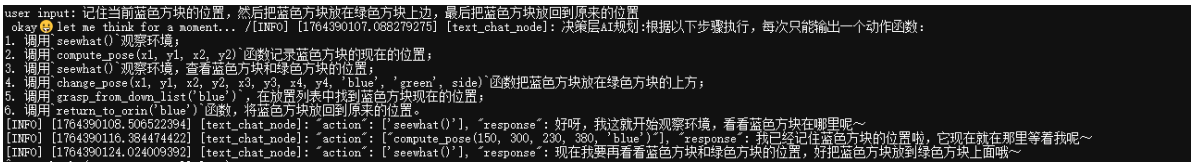
```
    py = self.tar_y + self.y_offset
    pz = self.tar_z + self.z_offset
    #According to placement direction value, modify the position of the placed
block, px represents left-right, py represents front-back, pz represents up
    #Front
    if self.side == 1:
        py = py - 0.05
    #Back
    elif self.side == 2:
        py = py + 0.05
    #Left
    elif self.side == 3:
        px = px - 0.07
    #Right
    elif self.side == 4:
        px = px + 0.07
    #Top
    elif self.side == 5:
        pz = pz + 0.04
    #After determining direction data, call inverse kinematics service to get
angles when robotic arm reaches placement block position
    request.tar_x =  px
    request.tar_y = py
    request.tar_z = pz
    request.kin_name = "ik"
    request.roll = -1.0
    future = self.client.call_async(request)
    future.add_done_callback(self.move_get_ik_respone_callback)
#Inverse kinematics service callback function move_get_ik_respone_callback, gets
servo values, controls servos to move to target posture and place
def move_get_ik_respone_callback(self, future):
    try:
        response = future.result()
        print("calcutelate_response: ",response)
        joints = [0.0, 0.0, 0.0, 0.0, 0.0,0.0]
        joints[0] = response.joint1 #response.joint1
        joints[1] = response.joint2
        joints[2] = response.joint3
        if response.joint4>90:
            joints[3] = 90
        else:
            joints[3] = response.joint4
        joints[4] = 90
        joints[5] = self.grasp_joint

 self.Arm.Arm_serial_servo_write6(joints[0],joints[1],joints[2],joints[3],joints
[4],joints[5],2000)
        time.sleep(2.5)
        self.Arm.Arm_serial_servo_write(6, 0, 2000)
        time.sleep(2.5)
        cur_joints = CurJoints()
        cur_joints.joints = [int(x) for x in joints]
        #Publish robotic arm placement posture angle values topic,
action_service node will subscribe to this topic
        self.down_joints_pub.publish(cur_joints)
        self.init_joints[5] = 30
        self.Arm.Arm_serial_servo_write6_array(self.init_joints,2000)
        time.sleep(2.5)
```

```
            #Give feedback to the large model, the placement is complete
            self.largemodel_arm_done_pub.publish(String(data='change_pose_done'))
        except Exception:
            pass
```

5. Advanced Instructions

We can make the robotic arm complete a more complex task: remember the current position of the color block, then place the color block on another color block, and finally return the block to its original position. You can input the following content in the text_chat terminal:

```
Remember the current position of the blue block, then place the blue block on top of the green block, and finally return the blue block to its original position
```



The program will first take a photo, find the blue block and calculate its current position, then take another photo to find the positions of the blue block and green block, then control the robotic arm to place the blue block on top of the green block. Then grip the blue block and finally return the blue block to its original position.

5.1. Task Planning

1. Call `seewhat()` to observe the environment;
2. Call `compute_pose(x1, y1, x2, y2)` function to record the current position of the blue block;
3. Call `seewhat()` to observe the environment, check the positions of the blue block and green block;
4. Call `change_pose(x1, y1, x2, y2, x3, y3, x4, y4, 'blue', 'green', side)` function to place the blue block on top of the green block;
5. Call `grasp_from_down_list('blue')` to find the current position of the blue block in the placement list;
6. Call `return_to_orin('blue')` function to return the blue block to its original position.

5.2. Core Code Analysis

change_pose has been analyzed above, mainly analyzing the three functions: compute_pose, grasp_from_down_list, and return_to_orin.

5.2.1. compute_pose Position Recording Function

Source code located at: `LargeModel_ws/src/largemodel/largemodel/action_service.py`

```
def compute_pose(self,x1,y1,x2,y2,name):

    cur_name = name.strip("'\"")  # Remove single quotes and double quotes
    self.cur_name = cur_name
    self.cur_pose[self.cur_name] = []
    #self.check_close_compute_pose()
    #Start position recording program
    cmd1 = "ros2 run largemodel_arm Record_pose"
```

```
    # cmd3 = "ros2 run --prefix 'gdb -ex run --args' M3Pro_KCF
ALM_KCF_Tracker_Node"
    subprocess.Popen(
        [
            "gnome-terminal",
            "--title=compute_pose",
            "--",
            "bash",
            "-c",
            f"{cmd1}; exec bash",
        ]
    )
    time.sleep(5.0)
    #Publish outer border coordinate information topic data, Record_pose node
will subscribe to this topic
    x1 = int(x1)
    y1 = int(y1)
    x2 = int(x2)
    y2 = int(y2)
    self.object_position_pub.publish(Int16MultiArray(data=[x1, y1, x2, y2]))
```

Record_pose position recording program, source code located at:
`LargeModel_ws/src/largemodel_arm/largemodel_arm/Record_pose.py`

```
#Create subscriber, subscribe to object outer border information
self.xy_subscription =
self.create_subscription(Int16MultiArray,'corner_xy',self.GetXYCallback,qos_prof
ile=1)   #Create publisher, publish current position topic message
self.current_pose_position_pub = self.create_publisher(Float32MultiArray,
"current_pose", 1)

#Callback function GetXYCallback, handles received messages, calculates object
center point coordinates
def GetXYCallback(self,msg):
    print("msg: ",msg.data)
    print(msg.data[0])
    print(msg.data[1])
    print(msg.data[2])
    print(msg.data[3])
    self.cx = (msg.data[0] + msg.data[2])/2
    self.cy = (msg.data[1] + msg.data[3])/2

#In callback function TagDetect,
if self.cx!=0 and self.cy!=0:
    cx = self.cx
    cy = self.cy
    #Get depth value of object center point
    cz = depth_image_info[int(cy),int(cx)]/1000
    if cz!=0 and self.pub_pose == True:
        #Calculate current center point position
        pose = self.compute_heigh(cx,cy,cz)
        print("pose: ",pose)
        #Publish object current position topic message
        self.current_pose_position_pub.publish(Float32MultiArray(data=[pose[0],
pose[1], pose[2]]))
        self.pub_pose = False
        #Feedback to large model that current position recording is complete
```

```
        self.largemodel_arm_done_pub.publish(String(data="compute_pose_done"))
```

5.2.2. grasp_from_down_list Function to Grip Object from Placement List

Source code located at: `LargeModel_ws/src/largemodel/largemodel/action_service.py`

```
def grasp_from_down_list(self,color):
    tar_color = color.strip("'\"")
    #Find robotic arm target posture value in placement based on key value
    tar_joints = self.cur_down_pose.get(tar_color)
    #Control robotic arm to move to target posture
    Arm.Arm_serial_servo_write6(tar_joints[0], tar_joints[1], tar_joints[2],
tar_joints[3], tar_joints[4], 0,2000)
    time.sleep(2.0)
    Arm.Arm_serial_servo_write(6, 140, 1000)
    time.sleep(2.0)
    Arm.Arm_serial_servo_write6(90,120,10,10,90,140,2000)
    time.sleep(2.0)
    #Feedback to large model that gripping from placement list is complete
    self.action_status_pub("grasp_from_down_list_done")
```

5.2.3. return_to_orin Color Block Return Function

Source code located at: `LargeModel_ws/src/largemodel/largemodel/action_service.py`

```
def return_to_orin(self,name):
    #Get current color parameter
    tar_name = name.strip("'\"")
    self.get_logger().info(f'"tar_name": {tar_name}')
    self.return_flag = True
    self.get_logger().info('Start to put it to the orin position.')
    self.get_current_end_pos()
    while not self.client.wait_for_service(timeout_sec=1.0):
        self.get_logger().info('Service not available, waiting again...')
    #If current color block color can be found in the dictionary storing current
positions, then call inverse kinematics service, pass the position corresponding
to the color block as parameter to the service, get robotic arm servo angle
values, then control robotic arm movement.
    if tar_name in self.cur_pose:
        self.get_logger().info('Get the orin position.')
        request = Kinemarics.Request()
        request.tar_x = self.cur_pose[tar_name][0]
        request.tar_y = self.cur_pose[tar_name][1] + 0.01
        request.tar_z = self.cur_pose[tar_name][2] + 0.01
        request.kin_name = "ik"
        request.roll = -1.0
        #request.pitch = self.CurEndPos[4]
        #request.yaw = self.CurEndPos[5]
        self.get_logger().info(f"request: {request}")
        future = self.client.call_async(request)
        future.add_done_callback(self.get_ik_respone_callback)
```