# Multimodal Large Model + Color Blocks Back In Place（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, first input the command to remember the current positions of color blocks in the terminal. After the program finishes recording, scramble the positions of the color blocks, then input the command to return the color blocks on the desktop to their original positions. After thinking, the large model will plan an action list, and the program will control the robotic arm to complete the color block return task according to the action functions in the action list.

## 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:

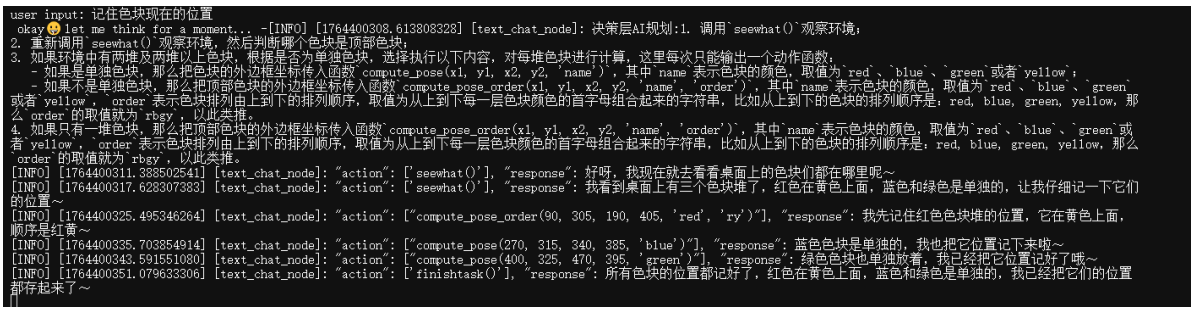
```
ros2 launch largemodel largemodel_control.launch.py text_chat_mode:=True
```

Then open a second terminal and input the following command:

```
ros2 run text_chat text_chat
```

Now input the command to remember the current positions of color blocks in the terminal. You can refer to the following example:

```
Remember the current positions of the color blocks
```




After recording the color block positions, input the end task command in the terminal:

```
End the position recording task
```

Then you can scramble the positions of the color blocks. After scrambling, input the color block return command in the terminal. You can refer to:

```
Return the blocks on the desktop to their original positions
```


After the buzzer beeps once, the program starts gripping color blocks and returning them to their original positions according to the order recorded in the program. If there are other color blocks above the current color block that needs to be returned, the upper color blocks will be removed first, then the target block will be gripped and returned to its original position.

# 3. Task Planning

## 3.1. Record Current Positions of Color Blocks

1. Call `seewhat()` to observe the environment;
2. Call `seewhat()` again to observe the environment, then determine which color block is the top block;
3. If there are two or more piles of color blocks in the environment, based on whether it's a single block, choose to execute the following content, calculate for each pile of blocks, here only one action function can be output at a time:
    - If it's a single block, pass the outer border coordinates of the block to the function `compute_pose(x1, y1, x2, y2, 'name')`, where `name` represents the color of the block, values are `red`, `blue`, `green` or `yellow`;
    - If it's not a single block, pass the outer border coordinates of the top block to the function `compute_pose_order(x1, y1, x2, y2, 'name', 'order')`, where `name` represents the color of the block, values are `red`, `blue`, `green` or `yellow`, `order` represents the stacking order from top to bottom, values are strings composed of the first letters of each layer's color from top to bottom, for example if the stacking order from top to bottom is: red, blue, green, yellow, then the value of `order` is `rbgy`, and so on.
4. If there's only one pile of blocks, pass the outer border coordinates of the top block to the function `compute_pose_order(x1, y1, x2, y2, 'name', 'order')`, where `name` represents the color of the block, values are `red`, `blue`, `green` or `yellow`, `order` represents the stacking order from top to bottom, values are strings composed of the first letters of each layer's color from top to bottom, for example if the stacking order from top to bottom is: red, blue, green, yellow, then the value of `order` is `rbgy`, and so on.

### 3.2. Color Blocks Return

1. First, call the `color_back_to_orin()` function to control color blocks to return to original positions.
2. Call the `check_remove(color)` function to check if the target color block exists in the removal list (where `color` is the color of the target block, value range is `red`, `blue`, `green` or `yellow`).
3. If the target color block exists in the removal list, execute the following operations:
   (1) Call the `grasp_from_rm_list(color)` function to grip the target color block from the removal list (meaning of `color` parameter is the same as above);
   (2) Pass the color of the target block as parameter to the `return_to_orin(color)` function, for example when the target block is red, call `return_to_orin(red)` (value range of `color` parameter is the same as above).
4. If the target color block does not exist in the removal list, follow these steps to return the block to its original position (only output one action function at a time):
   (1) Call the `seewhat()` function to find the current target block that needs to be returned;
   (2) Call the `seewhat()` function to determine if there are other objects above the current target block that needs to be returned;
   (3) If there are no other color blocks on top of the current target block that needs to be returned, execute the following steps:
   ① Call the `grasp_obj(x1, y1, x2, y2)` function to grip the current target block that needs to be returned (where `(x1, y1, x2, y2)` are the top surface border coordinates of the current target block that needs to be returned);
   ② Pass the color of the target block as parameter to the `return_to_orin(color)` function (value range of `color` parameter is the same as above).
   (4) If there are other color blocks on top of the current target block that needs to be returned, execute the following steps:
   ① Call the `remove_obj(x1, y1, x2, y2, color)` function to remove that block (where `(x1, y1, x2, y2)` are the top surface border coordinates of that block, `color` is the color of that block, value range is `red`, `blue`, `green` or `yellow`);
   ② Call the `seewhat()` function to observe the environment;
   ③ If there are no other color blocks above the current target block that needs to be returned, call the `grasp_obj(x1, y1, x2, y2)` function to grip the target block (where `(x1, y1, x2, y2)` are the top surface border coordinates of the target block);
   ④ Pass the color of the target block as parameter to the `return_to_orin(color)` function (value range of `color` parameter is the same as above)

## 4. Core Code Analysis

### 4.1. compute_pose_order Record Stacked Color Block Position

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

```
#Parameter description
#x1,y1,x2,y2: Top outer border coordinates of the top color block, x1,y1
represent
#name: Color of the top color block
#order: Represents the stacking order list from top to bottom, for example if the
stacked colors from top to bottom are red and yellow, then the passed order is
ry
def compute_pose_order(self,x1,y1,x2,y2,name,order):
    #Get current color block parameters
```

```
    cur_name = name.strip("'\"")
    color_order = order.strip("'\"")
    #Create a list to store the stacking order of color blocks from top to
bottom
    order_list = list(color_order[1:])
    self.get_logger().info(f"order_list:{order_list}")
    self.cur_name = cur_name
    self.cur_pose[self.cur_name] = []
    #Start position calculation program
    cmd1 = "ros2 run largemodel_arm Record_pose"
    proc = subprocess.Popen(
        [
            "gnome-terminal",
            "--title=compute_pose",
            "--",
            "bash",
            "-c",
            f"{cmd1}; exec bash",
        ]
    )
    self.pid_num = proc.pid
    self.get_logger().info(f"self.pid_num:{self.pid_num}")
    time.sleep(5.0)
    x1 = int(x1)
    y1 = int(y1)
    x2 = int(x2)
    y2 = int(y2)
    self.object_position_pub.publish(Int16MultiArray(data=[x1, y1, x2, y2]))

    while not self.compute_pose_future.done():
        if self.interrupt_flag:
            self.check_close_compute_pose()
            Arm.Arm_serial_servo_write6(90,150,12,20,90,30,1000)
            self.stop()
            return
        time.sleep(0.1)

    result = self.compute_pose_future.result()
    self.check_close_compute_pose()
    if not self.interrupt_flag:
        if result.data == "compute_pose_done":
            for i in range(0,len(order_list)):
                #Here we consider that xy directions are consistent with the top
block, only height changes, so just need to change z value, decreasing by 2.8cm
each time, approximately the height of one color block
                x = self.cur_pose[cur_name][0]
                y = self.cur_pose[cur_name][1]
                z = self.cur_pose[cur_name][2] - (i+1)*0.028
                if order_list[i] == 'r' or  order_list[i] == 'R':
                    self.cur_pose['red'] = []
                    self.cur_pose['red'].append(x)
                    self.cur_pose['red'].append(y)
                    self.cur_pose['red'].append(z)
                elif order_list[i] == 'b' or  order_list[i] == 'B':
                    self.cur_pose['blue'] = []
                    self.cur_pose['blue'].append(x)
                    self.cur_pose['blue'].append(y)
                    self.cur_pose['blue'].append(z)
```

```
                elif order_list[i] == 'y' or  order_list[i] == 'Y':
                    self.cur_pose['yellow'] = []
                    self.cur_pose['yellow'].append(x)
                    self.cur_pose['yellow'].append(y)
                    self.cur_pose['yellow'].append(z)
                elif order_list[i] == 'g' or  order_list[i] == 'G':
                    self.cur_pose['green'] = []
                    self.cur_pose['green'].append(x)
                    self.cur_pose['green'].append(y)
                    self.cur_pose['green'].append(z)

            self.get_logger().info(f"self.cur_pose:{self.cur_pose}")
            #Feedback to large model that recording stacked color block
positions is complete
            self.action_status_pub("compute_pose_order_done", x1=x1, y1=y1,
x2=x2, y2=y2,name=name,order=order)
        else:
            self.action_status_pub("compute_pose_failed", x1=x1, y1=y1, x2=x2,
y2=y2)
```

## 4.2. color_back_to_orin Publish Color Block Return Order

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

```
def color_back_to_orin(self):
    self.back_list = True
    #Buzzer beeps once
    Arm.Arm_Buzzer_On()
    time.sleep(0.5)
    Arm.Arm_Buzzer_Off()
    #Traverse the list storing color block positions in reverse order, feedback
to large model the current color block return instruction
    for key in reversed(self.cur_pose.keys()):
        if key == 'red':
            self.action_status_pub("send_red_back_to_orin")
        elif  key == 'green':
            self.action_status_pub("send_green_back_to_orin")
        elif key == 'blue':
            self.action_status_pub("send_blue_back_to_orin")
        elif key == 'yellow':
            self.action_status_pub("send_yellow_back_to_orin")
        while not self.return_done:
            time.sleep(0.1)
        self.return_done = False
    self.action_status_pub("return_to_orin_done")
    self.back_list = False
"send_red_back_to_orin": "Robot feedback: Return the red block to its original
position",
"send_green_back_to_orin": "Robot feedback: Return the green block to its
original position",
"send_blue_back_to_orin": "Robot feedback: Return the blue block to its original
position",
"send_yellow_back_to_orin": "Robot feedback: Return the yellow block to its
original position"
```

### 4.3. check_remove(color) Check if Current Target Color Block Exists in Removal List

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

```
def check_remove(self,color):
    #Get current color block color
    check_color = color.strip("'\"")
    #If current color block color can be found in removal list, then feedback to
large model that current color block can be found in removal list
    if check_color in self.cur_rm_pose:
        self.action_status_pub("color is in the rm_list")
    else:
        self.action_status_pub("color is not in the rm_list")

"color is in the rm_list" : "Color block exists in removal list, grip this color
block from removal list",
"color is not in the rm_list" : "Color block does not exist in removal list,
determine if there are other objects on top of this color block",
```

### 4.4. grasp_from_rm_list Grip Color Block from Removal List

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

```
def grasp_from_rm_list(self,color):
    #Get current color block color
    tar_color = color.strip("'\"")
    #Find corresponding robotic arm posture value in removal dictionary based on
current color
    tar_joints = self.cur_rm_pose.get(tar_color)
    Arm.Arm_serial_servo_write6(tar_joints[0], tar_joints[1], tar_joints[2],
tar_joints[3], tar_joints[4], tar_joints[5],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 removal list is complete
    self.action_status_pub("grasp_from_rm_list_done")
```