Multimodal Large Model + Robotic Arm Tracking（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 robotic arm tracking object commands through the terminal. According to the type of object to be tracked, the large model will start external programs to control the robotic arm to track the target object.

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
```

Robotic arm tracking has two types of objects:

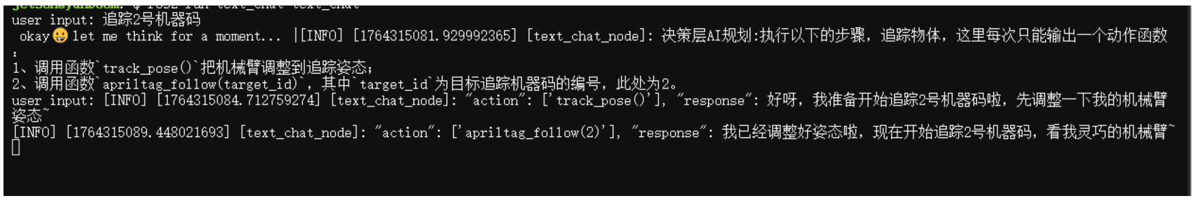
- Track machine code: Track machine code x, where x is the machine code ID, values are 1-4, for example, track machine code 1.
- Track other objects: Track any object, for example, track the small yellow duck on the green wooden block.

2.1. Track Machine Code

2.1.1. Startup

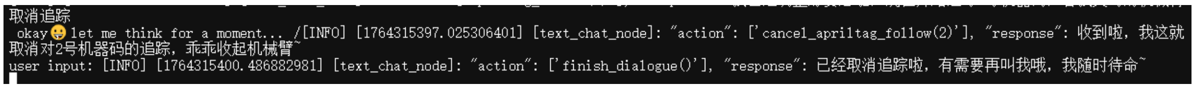
Input in the text_chat terminal:

```
Track machine code 2
```



```
user input: 追踪2号机器码
 okay😛let me think for a moment... |[INFO] [1764315081.929992365] [text_chat_node]: 决策层AI规划:执行以下的步骤，追踪物体，这里每次只能输出一个动作函数
:
1、调用函数`track_pose()`把机械臂调整到追踪姿态；
2、调用函数`apriltag_follow(target_id)`，其中`target_id`为目标追踪机器码的编号，此处为2。
user input: [INFO] [1764315084.712759274] [text_chat_node]: "action": ['track_pose()'], "response": 好呀，我准备开始追踪2号机器码啦，先调整一下我的机械臂姿态~
[INFO] [1764315089.448021693] [text_chat_node]: "action": ['apriltag_follow(2)'], "response": 我已经调整好姿态啦，现在开始追踪2号机器码，看我灵巧的机械臂~
```

First, the robotic arm will adjust to the tracking posture, then acquire the current image and start recognizing and tracking the target machine code. If the recognized machine code is not 2, the robotic arm will not track and will print "**Not target apriltag**" in the terminal; if the recognized machine code is 2, the robotic arm will start tracking, moving the machine code slowly, and the robotic arm will follow the movement. If you want to exit tracking, input in the text_chat terminal: **Cancel tracking**.



```
取消追踪
 okay😛let me think for a moment... /[INFO] [1764315397.025306401] [text_chat_node]: "action": ['cancel_apriltag_follow(2)'], "response": 收到啦，我这就
取消对2号机器码的追踪，乖乖收起机械臂~
user input: [INFO] [1764315400.486882981] [text_chat_node]: "action": ['finish_dialogue()'], "response": 已经取消追踪啦，有需要再叫我哦，我随时待命~
```

2.1.2. Task Planning

1. Call track_pose() function to adjust the robotic arm posture to tracking posture;
2. Call apriltag_follow(2) function, where the passed parameter is 2, representing the ID of the machine code to be tracked.

2.1.3. Core Code Analysis

track_pose() function, source code located at: LargeModel_ws/src/largemodel/largemodel/action_service.py

```
def track_pose(self):  # Robotic arm tracking posture
    #Directly communicate with the underlying control board, send values for 6 
robotic arm servos
    Arm.Arm_serial_servo_write6(90,150,12,20,90,30,1000)
    time.sleep(3.0)
    if not self.combination_mode and not self.interrupt_flag:
        self.action_status_pub("track_pose_done")
```

apriltag_follow function, source code located at: LargeModel_ws/src/largemodel/largemodel/action_service.py

```
def apriltag_follow(self,target_id):
    target_idf = float(target_id)
    #Open a terminal, run machine code tracking program, pass parameter 
target_id, representing the machine code ID to be tracked
    cmd1 = f"ros2 run largemodel_arm apriltag_follow_2D --ros-args -p 
target_id:={target_idf:.1f}"
    subprocess.Popen(
        [
            "gnome-terminal",
            "--title=apriltag_follow",
            "--",
            "bash",
            "-c",
            f"{cmd1}; exec bash",
        ]
    )
```

Machine code tracking program apriltag_follow_2D, source code path at: `LargeModel_ws/src/largemodel/largemodel/action_service.py`

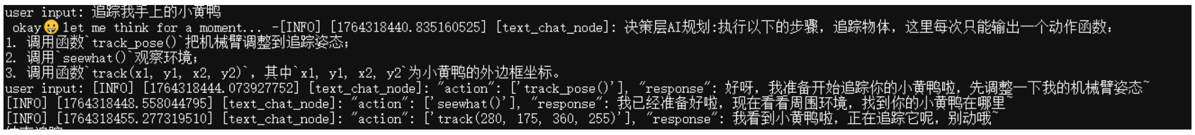
```
#Get target tracking id through command line parameters
self.declare_parameter('target_id', 0.0)
self.TargetID = 
int(self.get_parameter('target_id').get_parameter_value().double_value)
#In the image callback function callback
cur_id = tags[0].tag_id
#If the currently recognized id is the target tracking id
if cur_id == self.TargetID:
    #Get the center point coordinates of the machine code
    center_x, center_y = tags[0].center
    #Determine if the center point coordinates are within 10 pixels error of the
image center
    if (abs(center_x-320) >10 or abs(center_y-240)>10) and
self.XY_Track_flag==True:
        #Call XY_track function for tracking, passing the current center point
coordinates
        self.XY_track(center_x,center_y)
        print("Tracking")
        print("-------------------------------------")
    else:
        print("Not target apriltag.")
```

2.2. Track Other Objects

2.2.1. Startup

Input in the text_chat terminal:

```
Track the small yellow duck in hand
```



```
user input: 追踪我手上的小黄鸭
 okay😛let me think for a moment... -[INFO] [1764318440.835160525] [text_chat_node]: 决策层AI规划:执行以下的步骤，追踪物体，这里每次只能输出一个动作函数:
1. 调用函数`track_pose()`把机械臂调整到追踪姿态;
2. 调用`seewhat()`观察环境;
3. 调用函数`track(x1, y1, x2, y2)`，其中`x1, y1, x2, y2`为小黄鸭的外边框坐标。
user input: [INFO] [1764318444.073927752] [text_chat_node]: "action": ['track_pose()'], "response": 好呀，我准备开始追踪你的小黄鸭啦，先调整一下我的机械臂姿态~
[INFO] [1764318448.558044795] [text_chat_node]: "action": ['seewhat()'], "response": 我已经准备好啦，现在看看周围环境，找到你的小黄鸭在哪里~
[INFO] [1764318455.277319510] [text_chat_node]: "action": ['track(280, 175, 360, 255)'], "response": 我看到小黄鸭啦，正在追踪它呢，别动哦~
```

First, the robotic arm will adjust to the tracking posture, then acquire the current frame image to find the small yellow duck in hand, then control the robotic arm to track the small yellow duck. Slowly move the position of the small yellow duck, and the robotic arm will follow the movement. If you want to exit tracking, input in the text_chat terminal: **Cancel tracking**.

2.2.2. Task Planning

1. Call function `track_pose()` to adjust the robotic arm to tracking posture;
2. Call `seewhat()` to observe the environment and find the small duck in hand;
3. Call function `track(x1, y1, x2, y2)`, where `x1, y1, x2, y2` are the outer border coordinates of the small yellow duck.

2.2.3. Core Code Analysis

track_pose() and seehat() functions have been analyzed before and will not be repeated here. Let's mainly look at the track function, which source code is located at: LargeModel_ws/src/largemodel/largemodel/action_service.py

```
#x1,y1,x2,y2: Object outer border coordinates
def track(self, x1, y1, x2, y2):
    #Start KCF tracking program
    cmd1 = "ros2 run largemodel_arm KCF_Track_Move"
    #Start KCF tracking positioning program
    cmd2 = "ros2 run largemodel_arm ALM_KCF_Tracker"
    subprocess.Popen(
        [
            "gnome-terminal",
            "--title=KCF_track",
            "--",
            "bash",
            "-c",
            f"{cmd1}; exec bash",
        ]
    )
    subprocess.Popen(
        [
            "gnome-terminal",
            "--title=ALM_KCF_Tracker_Node",
            "--",
            "bash",
            "-c",
            f"{cmd2}; exec bash",
        ]
    )
    #Publish outer border coordinate information topic data, ALM_KCF_Tracker 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]))
```

KCF_Track_Move program, this program source code location: LargeModel_ws/src/largemodel_arm/largemodel_arm/KCF_Track_Move.py

```
#Import library, this library path is 
LargeModel_ws/src/largemodel_arm/largemodel_arm/Dofbot_Track.py
from largemodel_arm.Dofbot_Track import *

#Create robotic arm tracking gripping object
self.dofbot_tracker = DofbotTrack()

#Create subscriber, subscribe to object position information topic (xy)
self.sub_grasp_status = 
self.create_subscription(Position,"/pos_xy",self.TrackAndGrap,100)
#Topic callback function, handle received topic message data,
def TrackAndGrap(self,position):
    center_x, center_y = position.x,position.y
    #If the current object's center point coordinates are not within 8 pixels 
error range of the image center
    if abs(center_x-320) >8 or abs(center_y-240)>8 :
        #Call XY_track function for tracking, passing the current center point 
xy coordinates
        self.dofbot_tracker.XY_track(center_x,center_y)
```

```
#View XY_track function in Dofbot_Track.py, calculate values for servos 1-4 based 
on center point coordinates and PID parameters
def XY_track(self,center_x,center_y):
    self.px = center_x
    self.py = center_y
    if not (self.target_servox>=180 and center_x<=320 and self.a == 1 or 
self.target_servox<=0 and center_x>=320 and self.a == 1):
        if(self.a == 0):
            self.xservo_pid.SystemOutput = center_x
            if self.x_out_range == True:
                if self.target_servox<0:
                    self.target_servox = 0
                    self.xservo_pid.SetStepSignal(630)
                if self.target_servox>0:
                    self.target_servox = 180
                    self.xservo_pid.SetStepSignal(10)
                self.x_out_range = False
            else:
                self.xservo_pid.SetStepSignal(320)
                self.x_out_range = False

            self.xservo_pid.SetInertiaTime(0.01, 0.1)

            target_valuex = int(1500 + self.xservo_pid.SystemOutput)

            self.target_servox = int((target_valuex - 500) / 10) -10

            if self.target_servox > 180:
                self.x_out_range = True

            if self.target_servox < 0:
                self.x_out_range = True

    if not (self.target_servoy>=180 and center_y<=240 and self.b == 1 or
self.target_servoy<=0 and center_y>=240 and self.b == 1):
        if(self.b == 0):
            self.yservo_pid.SystemOutput = center_y

            if self.y_out_range == True:
                self.yservo_pid.SetStepSignal(450)
                self.y_out_range = False
            else:
                self.yservo_pid.SetStepSignal(240)

            self.yservo_pid.SetInertiaTime(0.01, 0.1)

            target_valuey = int(1500 + self.yservo_pid.SystemOutput)

            if target_valuey<=1000:
                target_valuey = 1000
                self.y_out_range = True
            self.target_servoy = int((target_valuey - 500) / 10) - 55
            if self.target_servoy > 180: self.target_servoy = 180
            if self.target_servoy < 0: self.target_servoy = 0

            joint2 = 120 + self.target_servoy
            joint3 =  self.target_servoy / 4.5
            joint4 =  self.target_servoy / 3
    joints_0 = [float(self.target_servox/1), float(joint2), float(joint3),
float(joint4), 90.0, 30.0]
    self.Arm.Arm_serial_servo_write6_array(joints_0,1000)
    self.cur_joints = joints_0
```