Multi module visual position application

1. Introduction

1.1 What is "Multimodal Visual Localization"?

Multimodal visual localization is a technology that combines multiple sensor inputs (such as cameras, depth sensors, and IMUs) with algorithmic processing techniques to accurately identify and track the position and posture of a device or user in an environment. This technology does not rely solely on a single type of sensor data, but instead integrates information from different perception modalities, thereby improving localization accuracy and robustness.

1.2 Overview of Implementation Principles

1. **Cross-modal Representation Learning**: In order for LLMs to process visual information, a mechanism must be developed to transform visual signals into a form that the model can understand. This may involve extracting features using convolutional neural networks (CNNs) or other architectures suitable for image processing and mapping them into the same embedding space as text.
2. **Joint Training**: By designing an appropriate loss function, text and visual data can be trained simultaneously within the same framework, allowing the model to learn to relate these two modalities. For example, in a question-answering system, an answer can be given based on both a text question and the associated image content. 3. Visually guided language generation/understanding: Once effective cross-modal representations are established, visual information can be leveraged to enhance the capabilities of language models. For example, given a photo, the model can not only describe what is happening in the image, but also answer specific questions about the scene and even execute instructions based on visual cues (e.g., navigating to a location).

2. Code Analysis

Key Code

1. Tool Layer Entry (`largemodel/utils/tools_manager.py`)

The `visual_positioning` function in this file defines the tool's execution flow, specifically how it constructs a prompt containing the target object name and formatting requirements.

```
# From largemodel/utils/tools_manager.py
class ToolsManager:
    # ...
    def visual_positioning(self, args):
        """
        Locate object coordinates in image and save results to MD file.
        定位图像中物体坐标并将结果保存为MD文件。

        :param args: Arguments containing image path and object name.
        :return: Dictionary with file path and coordinate data.
        """
        self.node.get_logger().info(f"Executing visual_positioning() tool with
args: {args}")
        try:
            image_path = args.get("image_path")
            object_name = args.get("object_name")
            # ... (Path fallback mechanism and parameter checking)

            # Construct a prompt asking the large model to identify the
coordinates of the specified object. / 构造提示，要求大模型识别指定物品的坐标。
            if self.node.language == 'zh':
                prompt = f"请仔细分析这张图片，用一个个框定位图像每一个{object_name}的位
置..."
            else:
                prompt = f"Please carefully analyze this image and find the
position of all {object_name}..."

            # ... (Building an independent message context)

            result = self.node.model_client.infer_with_image(image_path, prompt,
message=message_to_use)

            # ... (Process and parse the returned coordinate text)

            return {
                "file_path": md_file_path,
                "coordinates_content": coordinates_content,
                "explanation_content": explanation_content
            }
        # ... (Error Handling)
```

2. Model Interface Layer (`largemodel/utils/large_model_interface.py`)

The `infer_with_image` function in this file serves as the unified entry point for all image-related tasks.

```
# From largemodel/utils/large_model_interface.py
```

```
class model_interface:
    # ...
    def infer_with_image(self, image_path, text=None, message=None):
        """Unified image inference interface. / 统一的图像推理接口。"""
        # ... (Prepare Message)
        try:
            # Determine which specific implementation to call based on the value
of self.llm_platform
            if self.llm_platform == 'ollama':
                response_content = self.ollama_infer(self.messages,
image_path=image_path)
            elif self.llm_platform == 'tongyi':
                # ... Logic for calling the Tongyi model
                pass
            # ... (Logic of other platforms)
        # ...
        return {'response': response_content, 'messages': self.messages.copy()}
```

Code Analysis

The core of the visual positioning function lies in **guiding large models to output structured data through precise instructions**. It also follows the layered design of the tool layer and the model interface layer.

1. **Tools Layer (`tools_manager.py`):**

- The `visual_positioning` function is the core of this function. It accepts two key parameters: `image_path` (the image path) and `object_name` (the name of the object to be positioned).
- The core operation of this function is **building a highly customized prompt**. It doesn't simply ask the model to describe an image. Instead, it embeds `object_name` into a carefully designed template, explicitly instructing the model to "locate each {object_name} in the image," and implicitly or explicitly requires the results to be returned in a specific format (such as an array of coordinates).
- After building the prompt, it calls the `infer_with_image` method of the model interface layer, passing the image and this customized instruction. * After receiving the returned text from the model interface layer, it needs to perform **post-processing**: using methods such as regular expressions to parse the model's natural language response to extract precise coordinate data.
- Finally, it returns the parsed structured coordinate data to the upper-layer application.

2. **Model Interface Layer (`large_model_interface.py`):**

- The `infer_with_image` function still serves as the "dispatching center." It receives the image and prompt from `visual_positioning` and dispatches the task to the correct backend model implementation based on the current configuration (`self.llm_platform`).
- For visual positioning tasks, the model interface layer's responsibilities are essentially the same as for visual understanding tasks: correctly packaging the image data and text instructions, sending them to the selected model platform, and then returning the returned text results intact to the tool layer. All platform-specific implementation details are encapsulated in this layer.

In summary, the general workflow for visual localization is: ToolsManager receives the target object name and constructs a precise prompt requesting coordinates. ToolsManager calls the model interface. ModelInterface packages the image and prompt together and sends them to the corresponding model platform according to the configuration. The model returns text containing the coordinates. ModelInterface returns this text to ToolsManager. ToolsManager parses the text, extracts the structured coordinate data, and returns it. This process demonstrates how Prompt Engineering can enable a general-purpose large visual model to accomplish more specific and structured tasks.

3.Practical Operation

3.1 Configuring Online LLM

1. **First, obtain your API key from any platform mentioned in the previous tutorial**
2. **Next, you'll need to update the key in the configuration file. Open the model interface configuration file** `large_model_interface.yaml`:

```
vim ~/yahboom_ws/src/largemodel/config/large_model_interface.yaml
```

3. **Fill in your API Key**:
   Find the corresponding part and paste the API Key you just copied into it. Here we take Tongyi Qianwen configuration as an example

```
# large_model_interface.yaml

## Thousand Questions on Tongyi
qianwen_api_key: "sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" # Paste your Key
qianwen_model: "qwen-vl-max-latest" #  You can choose the model as needed,
such as qwen-turbo, qwen-plus
```

4. **Open the main configuration file** `yahboom.yaml`:

```
vim ~/yahboom_ws/src/largemodel/config/yahboom.yaml
```

5. **Select the online platform you want to use**:
   Modify the `llm_platform` parameter to the platform name you want to use

```
# yahboom.yaml

model_service:
  ros__parameters:
    # ...
    llm_platform: 'tongyi'  #Optional Platform: 'ollama', 'tongyi', 'spark',
'qianfan', 'openrouter'
```

3.2 Start and test the functionality

1. **start up**:

   Note: The startup commands for CSi cameras and USB microphone cameras are different. Please run the appropriate command for your camera.

   CSI Camera

Starting UDP Video Streaming (host machine)

```
./start_csi.sh
```

Enter the CSI camera docker (host machine)

```
./run_csi_docker.sh
```

Start topic conversion (docker)

```
python3 ~/temp/udp_camera_publisher.py
```

View container id

```
docker ps
```

According to the container ID shown above, multiple terminals enter the same docker

```
docker exec -it  container_id  /bin/bash
```

Run the following command to enable voice interaction:

```
ros2 launch largemodel largemodel_control.launch.py
```

USB Camera

Enter the USB camera docker (host machine)

```
./run_usb_docker.sh
```

Run the following command to enable voice interaction:

```
ros2 launch largemodel largemodel_control.launch.py
```

2. **Test**:
   - **Wake up**: Say "Hi, yahboom" into the microphone.
   - **Dialogue**: After the speaker responds, you can say, "Analyze the position of the dinosaur in the image."
   - **Observe the log**: In the terminal running the `launch` file, you should see the following:
   1. The ASR node recognizes your question and prints it out.
   2. The `model_service` node receives the text, calls the LLM, and prints the LLM's response.
   - **Listen for the response**: After a while, you should hear the response from the speaker and find an md file in the `/root/yahboom_ws/src/largemodel/resources_file/visual_positioning` path that records the coordinate position and positioned object information.
3. **FAQ**:

Change to your own pictures

(1) Rename the image to test_image.jpg and place it in the ~/temp directory for later use.

(2) Enter any Docker terminal

```
cd ~/temp
```

```
cp test_image.jpg
~/yahboom_ws/src/largemodel/resources_file/visual_positioning
```

Copy the image to the ~/yahboom_ws/src/largemodel/resources_file/visual_positioning directory

(3) Restart the `largemodel` main program