2024.5.1

基于大模型提示的任务规划迁移实践

一、前言：

在机器人操作任务中，确保任务完成的首要因素就是要有合理的任务规划，但是对于很多相似的操作任务来说，每次重新进行任务规划得到的结果都是相似的，因此能否寻找一种方式，把机器人操作任务的任务规划能够在相似的场景之间进行迁移就成为了我们感兴趣的问题。

对于传统的操作任务的任务规划来说，首先要确定的就是根据任务场景的状态和智能体自身（主要是机器人）的能力合理规划出要操作的目标和先后次序，这需要智能体对场景具有一定的理解能力。这种理解能力的获得可以通过视觉信息获取，如视觉的rgb图像和点云图像，也可以通过常识知识获取，如搜索相应的知识库（如互联网），获取当前操作物体的特性。在本研究实践中，我们默认已经通过技术手段获取了任务场景中各物体的属性知识，需要完成相似但不同场景之间的任务规划。

我们借助大模型带来的语义理解能力和推理能力，完成任务规划的迁移，为此我们尝试构建一套prompt范式，确保我们任务迁移的成功。

二、方法方式：

第一步：预准备阶段：描述想要LLM完成的任务，并让LLM复述你的想法，看是否准确：

I will give you a manipulation task description and I need you help me to finish the task planning. The task description consists of three parts: The main task, the reference task, and the scene description.

The main task and the reference task have the same structure, the reference task's planning has been known while the main task's planning is not known. Each task consists of several sub-tasks, and each sub-task consists of three parts: "action", "require\_states" and "obtain\_states". "require\_states" are initial states before"action" and "obtain\_states" are final states after "action". By adopting this structure, task planning can reflect changes in the scene.

The scene description and task planning are knowledge graphs written by py2neo and the original XML format, which include all needed knowledge to be used in task planning, such as objects and their parts to be manipulated, and the task planning should use the object in the scene description.

The aim of task planning is that when the reference task and the scene description have been known, and I give a new task named the main task, you can give me a reasonable task planning result.

Do you understand? If yes, re-say what I mean detailedly.

第二步：依次描述操作场景，使用代码描述。代码格式越统一，LLM理解起来越方便：

The following is the basic scene description, which reflects the relationships between the object and its sub-objects:

## The Python code about knowledge graph of scene description

object\_1 = Node('object', name='cabinet')

object\_1\_1 = Node('sub\_object', name='base', mass='0.35')

object\_1\_2 = Node('sub\_sub\_object', name='drawer\_link', mass='7.85398', diaginertia="0.923301 0.764585 0.168533")

object\_1\_3 = Node('sub\_sub\_sub\_object', name='cabinet\_handle', mass='0.18')

skill\_graph.create(object\_1)

skill\_graph.create(object\_1\_1)

skill\_graph.create(object\_1\_2)

skill\_graph.create(object\_1\_3)

relationship2\_1 = Relationship(object\_1\_1, 'part', object\_1)

relationship3\_1 = Relationship(object\_1\_2, 'part', object\_1)

relationship4\_1 = Relationship(object\_1\_3, 'part', object\_1)

relationship2\_2 = Relationship(object\_1\_2, 'relative position', object\_1\_1, connection\_type='joint',

joint\_type='slide',

joint\_range='-0.1 0', joint\_axis='0 1 0', joint\_pos='0 0 0',

joint\_damping='100.0', pos='0 -0.01 0.076', quat='1 0 0 0', inertial\_pos='0 0 0.35',

inertial\_quat='0.5 0.5 0.5 0.5')

relationship2\_3 = Relationship(object\_1\_3, 'relative position', object\_1\_2, connection\_type='attach', pos='0 -0.16 0.04', quat='1 0 0 0',

inertial\_pos='0 0 0 0', inertial\_quat='1 0 0 0')

skill\_graph.create(relationship2\_1)

skill\_graph.create(relationship3\_1)

skill\_graph.create(relationship4\_1)

skill\_graph.create(relationship2\_2)

skill\_graph.create(relationship2\_3)

object\_2 = Node('object', name='door\_lock')

object\_2\_1 = Node('sub\_object', name='frame', mass='7.85398')

object\_2\_2 = Node('sub\_sub\_object', name='door', mass='2.43455', diaginertia='0.0913751 0.0521615 0.043714')

object\_2\_3 = Node('sub\_sub\_sub\_object', name='latch', mass='0.1', diaginertia='0.0483771 0.0410001 0.0111013')

skill\_graph.create(object\_2)

skill\_graph.create(object\_2\_1)

skill\_graph.create(object\_2\_2)

skill\_graph.create(object\_2\_3)

relationship2\_1 = Relationship(object\_2\_1, 'part', object\_2)

relationship3\_1 = Relationship(object\_2\_2, 'part', object\_2)

relationship4\_1 = Relationship(object\_2\_3, 'part', object\_2)

relationship3\_2 = Relationship(object\_2\_2, 'relative position', object\_2\_1, connection\_type='joint',

joint\_type='hinge',

joint\_range="0.0 0.4", joint\_axis="0 0 1", joint\_pos='0.255 0 0',

joint\_damping='1', pos='0.3 0 0', quat='1 0 0 0', inertial\_pos='0.0296816 -0.00152345 0',

inertial\_quat='0.701072 0 0 0.713091')

relationship3\_3 = Relationship(object\_2\_3, 'relative position', object\_2\_2, connection\_type='attach', pos='-0.175 0 -0.025', quat='1 0 0 0',

inertial\_pos='-0.017762 0.0138544 0',

inertial\_quat='0.365653 0.605347 -0.36522 0.605365')

skill\_graph.create(relationship2\_1)

skill\_graph.create(relationship3\_1)

skill\_graph.create(relationship4\_1)

skill\_graph.create(relationship3\_2)

skill\_graph.create(relationship3\_3)

第三步：描述参考任务规划，需要注意的是任务规划我们采用的形式既不是代码也不是语言描述，而是我们自己定义的一套表达形式，因此在输入完整的任务规划前，要先详细描述这套表达形式的含义：

before starting the task planning, I would like to give you more information and rules about the task planning description.

As for the relationships, we assume that there exists five connection types: attach, joint, in, on, and around. The default connection type is "attach". In the task planning, we use the "object -> connection types -> object" to describe this relationship. We use the form: "object(property)" to present the object and its properties, and as for relationships, we use the form "connection type(property)" to present the relationship and its property.

for example:

base(mass='0.35')->joint(joint\_damping='100.0')->drawer link(mass='7.85')

If you understand, use the form "object(property) -> connection types(property) -> object(property)" to present the objects/sub\_objects and their properties.

在LLM复述正确后，给出其完整的参考任务规划。

第四步：参考任务规划，并给出主任务要求，完成任务规划的迁移。在这一步中，约束/规定”notes”是很重要的，这表明我们希望任务规划生成哪些信息、在哪里选择参考的信息、不希望生成何种信息等。

Great! Now I will give you the reference task and its planning.

reference task: "open the cabinet", the target object is "cabinet".

task planning:

{"task": open the cabinet}

{"sub\_task":1, "action": "move", "require\_states": gripper(open), "obtain\_states":gripper(open)-> around -> cabinet handle}

{"sub\_task":2, "action":"pick", "require\_states": gripper(open) -> around -> cabinet handle, "obtain\_states": gripper(closed) -> around -> cabinet handle}

{"sub\_task":3, "action":"move", "require\_states": gripper(closed) -> around -> Cabinet Handle && base -> joint(joint\_range=0) -> drawer link, "obtain\_states":gripper(closed) -> around -> Cabinet Handle and base -> joint(joint\_range=-0.1) -> drawer link}

{"sub\_task":4, "action":"drop", "require\_states": gripper(closed), "obtain\_states": gripper(open)}

Notes: The "gripper" is the manipulator, which can be seen as an object. It has two states: open and closed. sub-task 1~4 is the order of the sub-task.

In task planning, the "action" can be chosen from "move", "pick" and "drop", and the sub\_objects that exist in the task planning should be part of the target object.

The notes should be followed in the task planning.

My question is: the main task is "open the door lock", and the target object is "door lock". Give me the task planning of the main task based on the knowledge of the reference task and scene description.

在LLM每次生成回答后，都要检查生成的回答是否准确，如果出现不准确的情况，就要指出并纠正。这是一个不断反馈-更新的过程。