Appendix

Contents

A1 Robot System Setup

A1.1 Robot hardware

Our robot is a centaur-like robot platform. The upper body of the robot is humanoid in design and is similar in size to the average human to adapt to both dual-arm and single-arm manipulation. The robot's mobility relies on its quadrupedal lower body and maintains whole-body balance to cope with a variety of terrain conditions and perform loco-manipulation tasks. Moreover, to improve the robot's mobility on flat ground, wheel modules are integrated underneath each leg and can control the direction and steering of the wheels.

The robot's whole body consists of 38 actuatable joints. The robot's torso is mounted on the pelvis of the lower body via yaw joints, allowing the upper body to rotate in the transverse plane. Each arm of the robot includes 6 DoF, where the right hand gripper contains one extra DoF that controls its opening and closing. The robot's legs are designed to provide an omni-directional wheeled motion and articulated legged locomotion, with each leg containing six degrees of freedom, allowing for positioning, orientation, and rotation of the wheeled-leg module.

The perception system of the robot consists of two on-board RealSense Depth Camera D435i, one located in the robot's head and the other in the robot's pelvis, which are used to provide 2D images and depth information of the surrounding environment and objects. The complete computing system consists of two on-board computing units (ZOTAC-EN1070K PC, COM Express conga-TS170) for system communication and real-time robot control and an external pilot PC (Inter Core i9-13900HX CPU @3.90GHz, NVIDIA GeForce RTX 4090) for task planning and sensory data processing as well as a user interface.

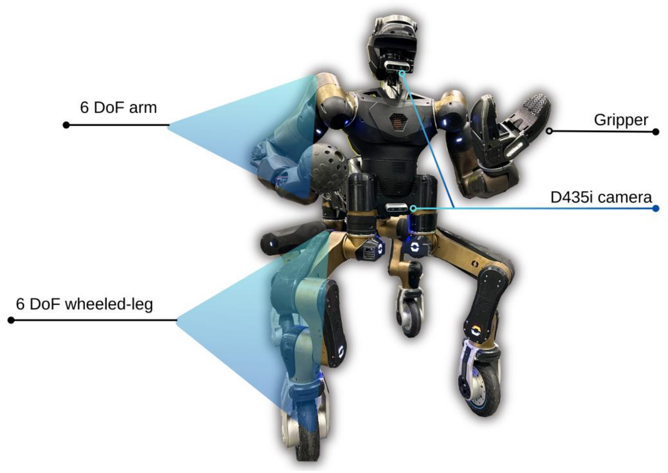


Figure A1: Robot hardware setup

A1.2 Robot software

We use XBotCore, a cross-platform, real-time, open-source software designed for interfacing with low-level hardware components of robots [1]. This innovative tool enables effortless programming and management of various robotic systems by offering a standardized interface that conceals the intricacies of the hardware. Additionally, a proprietary CartesI/O motion controller [2] handles higher-order motion instructions. It is capable of managing multiple responsibilities and restrictions, prioritized according to the demands of specific situations. Through solving a series of quadratic programming (QP) challenges, each linked to a unique priority tier, the controller ensures optimal performance across all preceding priority stages.

A2 Details of Robot Learning

We utilize Proximal Policy Optimization (PPO) [3] for training our tasks, employing a multi-layer perceptron within an actor-critic framework. The network architecture for the drawer opening, door opening, and dual-arm picking tasks consists of layers with [256, 128, 64] units while the picking task uses layers with [256, 128, 64] units. The activation function applied across all tasks is ELU. Below, we detail the observations, task-specific rewards (r_{task}), and reward parameters for each task.

A2.1 drawer opening

First, we define the frame of the drawer handle. The x-axis of the handle points towards the robot, while the z-axis points upwards. The handle's inward direction is aligned negatively along the x-axis, and the upward direction is consistent with the z-axis. The task reward is defined as

$$r_{task} = \alpha_7 r_{around} + l_{drawer} * r_{around} + l_{drawer} \tag{A1}$$

where $r_{around} = 0.5$ when the gripper's top link is above the handle's position and the bottom link is below the handle's position, otherwise $r_{around} = 0$. l_{drawer} represents the length by which the drawer has been pulled.

The observations and reward parameters for this task are listed in Tab. 1 and 2.

normalized upper body joints position
upper body joints velocity * 0.1
drawer pulled length
vector from gripper to drawer handle

Table 1: observations of drawer opening task

α_1	2.0
α_2	0.0
α_3	0.5
α_4	7.5
α_5	7.5
α_6	0.01
α_7	0.7
β	0.04

Table 2: reward parameters of drawer opening task

A2.2 door opening

The door handle has the same frame as the drawer handle. The task reward is defined as

$$r_{task} = \alpha_7 r_{around} + angle_{handle} * r_{around} + angle_{handle} + angle_{door} \tag{A2}$$

where r_{around} is the same setting as the drawer opening task and $angle_{handle}$ represents the angle by which the door handle has been pushed. $angle_{door}$ is the angle of the opened door.

The observations and reward parameters for this task are listed in Tab. 3 and 4.

base pose
right arm joints position
door handle pose
gripper pose
door handle angle
door opened angle

Table 3: observations

α_1	2.0
α_2	0.0
α_3	1.5
α_4	7.5
α_5	2.0
α_6	0.01
α_7	0.125
β	0.02

Table 4: parameters

A2.3 single arm picking

We define the object's upward direction as aligning negatively along the x-axis, and the inward direction as aligning negatively along the z-axis. This orientation encourages the gripper to adopt a top-to-bottom pose, facilitating a proper grasp of the object. The task reward is defined as

$$r_{task} = \alpha_7 r_{around} + h \tag{A3}$$

where r_{around} is the same setting as the previous tasks with the corresponding object frame and $h = 1$ if the object is been picked up, otherwise $h = 0$.

The observations and reward parameters for this task are listed in Tab. 5 and 6.

base pose
right arm joints position
object pose
gripper pose

Table 5: observations

α_1	7.5
α_2	0.0
α_3	5.0
α_4	2.5
α_5	7.5
α_6	0.01
α_7	0.7
β	0.1

Table 6: parameters

A2.4 dual arm picking

In the dual arm picking task, the distance d_l and d_r represents the left end-effector and right end-effector to the left and right side of the object, respectively. The task reward is defined as

$$r_{task} = h \tag{A4}$$

where $h = 1$ if the object is been picked up, otherwise $h = 0$.

The observations and reward parameters for this task are listed in Tab. 7 and 8.

base pose
two arms joints position
object pose
left end-effector pose
right end-effector pose
vector from object left side to left end-effector
vector from object right side to right end-effector

α_1	2.0
α_2	2.0
α_3	0.0
α_4	0.0
α_5	7.5
α_6	0.01
α_7	0.0
β	0.0

Table 8: parameters

Table 7: observations

A3 Details of Whole-body Optimization

The trajectory optimization problem essentially constitutes a Nonlinear Programming (NLP) challenge characterized by a predetermined quantity of nodes and intervals. Its canonical formulation typically adheres to Eq.(A5)

$$\begin{cases} \min_{\mathbf{x}(.),\mathbf{u}(.)} \int_0^T L(\mathbf{x}(t), \mathbf{u}(t), t)dt \\ \text{s.t. } \dot{\mathbf{x}}(t) = \boldsymbol{f}(\mathbf{x}(t), \mathbf{u}(t), t) \\ \boldsymbol{g}_1(\mathbf{x}(t), \mathbf{u}(t), t) = 0 \\ \boldsymbol{g}_2(\mathbf{x}(t), \mathbf{u}(t), t) \leq 0 \end{cases} \tag{A5}$$

the standard formulation necessitates conversion into a discrete programming format . Subsequently, we discrete the state and input variable as the follow sets, N is the node number

$$\mathcal{X} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \end{bmatrix}; \mathcal{U} = \begin{bmatrix} \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_N \end{bmatrix} \tag{A6}$$

then the general optimization form Eq.(A5) becomes Eq.(A7)

$$\begin{gathered} J = \sum_{i=0}^{N} L_i(\mathbf{x}_i, \mathbf{u}_i) \\ \dot{\mathbf{x}}_i = \boldsymbol{f}(\mathbf{x}_i, \mathbf{u}_i)\ , i = 0, \cdots N \\ \mathbf{C}_{\min} \leq \mathbf{C}(\mathbf{x}_i, \mathbf{u}_i) \leq \mathbf{C}_{\max}, i = 0, \cdots N \end{gathered} \tag{A7}$$

where , $\mathbf{C}(\mathbf{x}_i, \mathbf{u}_i)$ is the discrete form of equality and inequality constrain, $\mathbf{C}_{\min}$ is the lower limit, $\mathbf{C}_{\max}$ is the upper limit. Specifically, in order to keep the trajectory feasible, we should shape the constrains as:

$$\begin{aligned} \mathbf{q}^0 &= \mathbf{q}_{\text{init}} && \text{initial position} \\ \mathbf{v}^0 &= 0 && \text{initial velocity} \\ \mathbf{q}^k_{\text{min}} &\leq \mathbf{q}^k \leq \mathbf{q}^k_{\text{max}} && \text{position bounds } \forall k \in [1, N-1] \\ \mathbf{v}^k &\leq \mathbf{v}^k \leq \mathbf{v}^k_{\text{max}} && \text{velocity bounds} \quad \forall k \in [1, N-1] \\ \dot{\mathbf{v}}^k_{\text{min}} &\leq \dot{\mathbf{v}}^k \leq \dot{\mathbf{v}}^k_{\text{max}} && \text{acceleration bounds} \quad \forall k \in [0, N-1] \\ \mathbf{f}^{z,k}_{c,i} \cdot \mathbf{n}_i > 0, \left\| (\mathbf{f}^{x,k}_{c,i}, \mathbf{f}^{y,k}_{c,i}) \right\|_2 &\leq \mu_i \left(\mathbf{f}^{z,k}_{c,i} \cdot \mathbf{n}_i \right) && \text{leg contact force bounds} \quad \forall k \in [0, N-1] \end{aligned} \tag{A8}$$

where $\mathbf{f}_{c,i} = [\mathbf{f}^x_{c,i}, \mathbf{f}^y_{c,i}, \mathbf{f}^z_{c,i}]$ is the i-th leg contact force. At the end of programming, its function of the whole body trajectory is to realize the motion learned from RL framework, we implement the cost as :

$$L_i(\mathbf{x}_i, \mathbf{u}_i) = \|\mathbf{q}^u_i - \mathbf{q}^*_i\|^2 + \|\mathbf{u}\|^2 \tag{A9}$$

the term $\|\mathbf{q}^u_i - \mathbf{q}^*_i\|^2$ is for merging the gap between RL trajectory and actually feasible trajectroy, $\mathbf{q}^u_i$ is the upper body trajectory from RL, $\mathbf{q}^*_i$ is the upper body trajectory from whole body optimization, $\|\mathbf{u}\|^2$ for reduce the energy of the whole motion.

A4 Motion Library

We constructed a motion library to house the learned whole-body skills as well as the action and condition nodes used to construct the task graph. The motion library includes information about the skills fed to the LLM, as well as the control code corresponding to each skill. The following Fig. A2, A3 shows the action skills and nodes inside the motion library that LLM can choose to invoke to construct the task graph.

```
### Action Node ###

<HomingPose>: 'name'='homing_pose'; 'type'=general; 'label'=start the robot to a initial position;
'description'=control the robot to power up and back to the initial robot pose.

<FindObject>: 'name'='find_object'; 'type'=general; 'label'=look around for object;
'description'=control the robot to turn on the head camera, and rotates itself to find 'object' and
acquire its 3D position.

<MoveTarget>: 'name'='move_target'; 'type'=wheel; 'label'=approach to target with wheels;
'description'=control the robot to approach to the target location using wheel motion (require knowing
3D position of 'target').

<WalkTarget>: 'name'='walk_target' ;'type'=leg; 'label'=approach to target with legs;
'description'=control the robot to approach to the target location using leg motion (require knowing
3D position of 'target').

<ObjectDetect>: 'name'='object_detect'; 'type'=general; 'label'=object detection and pose estimation;
'description'=using the head camera to detect and estimate the position and pose of the
'target_object'.

<ObjectPlace>: 'name'='object_place'; 'type'=general; 'label'=place object to a target position;
'description'=control the robot to put the object to a target position. (require knowing 'target' 3D
position).

<OpenDoor>: 'name'='door_open'; 'type'=general; 'label'=open the door; 'description'=control the robot
to open the door. (require knowing 3D position of 'door').

<SinglePick>: 'name'='single_arm_pick'; 'type'=single_arm; 'label'=grasp object and pick it up;
'description'=control the robot to grasp the target object with right arm, and pick it up (require
knowing 'target_object' position and pose).

<DualPick>: 'name'='dual_arm_pick'; 'type'=dual_arm; 'label'=hold object with dual arms and pick it
up. 'description'=control the robot to hold the target object with dual arms, and pick it up (require
knowing 'target_object' position and pose).

<OpenDrawer>: 'name'='open_drawer'; 'type'=general; 'label'=open the drawer. 'description'=control the
robot to open the drawer. (require knowing 3D position of 'drawer').
```

Figure A2: Action nodes in the motion library, where the blue nodes are based on learned whole-body motion skills.

```
### Condition Node ###

<Distance>: 'name'='object_in_reach'; 'type'=general; 'label'=is object in reach;
'description'=measure the distance from the object to the robot. if it is larger than 80cm then return
to fail. (require knowing 'object' 3D position)

<WhetherSingleArm>: 'name'='whether_single_arm'; 'type'=general; 'label'=select robotic morphology
based on manipulation task; 'description'=apply VLM to reason whether to use a single arm or dual arm
to manipulate object, can be used to make decisions before picking actions.

<WhetherWheelMove>: 'name'='whether_wheel_move'; 'type'=general; 'label'=select robotic morphology
based on locomotion task; 'description'=apply VLM to reason whether to use wheel or leg to move, can
be used to make decisions before locomotion actions.

<IsActionSuccess>: 'name'='is_action_completed': 'type'=general; 'label'=reason about the success of
the action; 'description'=apply VLM to reason whether the previous manipulation action is successful,
if not, repeat the action one time in behavior tree.
```

Figure A3: Condition nodes with different functions in the motion library.

A5 Motion Morphology Selection

In this section, we show the task scenarios used for the motion morphology selection experiments.

A5.1 Manipulation Scenarios

For the robot manipulation morphology selection experiments included six simulated and four real-world scenarios. We conducted ten morphology selections for each scenario, and before each trial, the positions and poses of the objects in the scenarios were reset. We applied the same prompts for all manipulation morphology selections, with the instructions for each scenario shown in Fig. A4.

A5.2 Locomotion Scenarios

The robot locomotion morphology selection experiments included six simulated and four real-world scenarios, as shown in Fig.A5. We conducted ten morphology selections for each scenario, and before each trial, the positions of the robot and obstacles in the scenarios were reset. We applied the same prompts for all locomotion morphology selections.

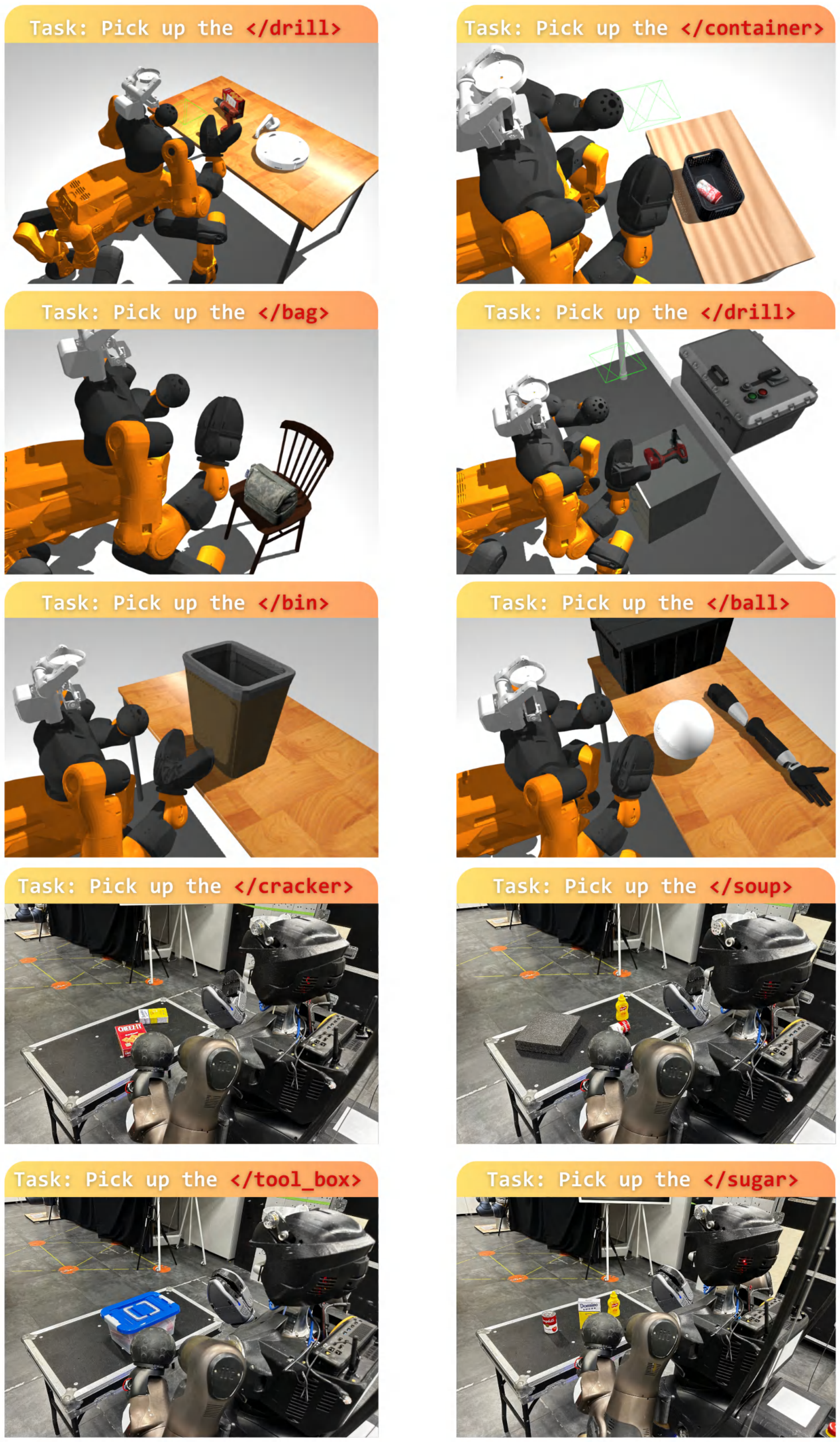

Task: Pick up the </drill>
Task: Pick up the </container>
Task: Pick up the </bag>
Task: Pick up the </drill>
Task: Pick up the </bin>
Task: Pick up the </ball>
Task: Pick up the </cracker>
Task: Pick up the </soup>
Task: Pick up the </tool_box>
Task: Pick up the </sugar>

Figure A4: Task scenarios for manipulation morphology selection experiments.

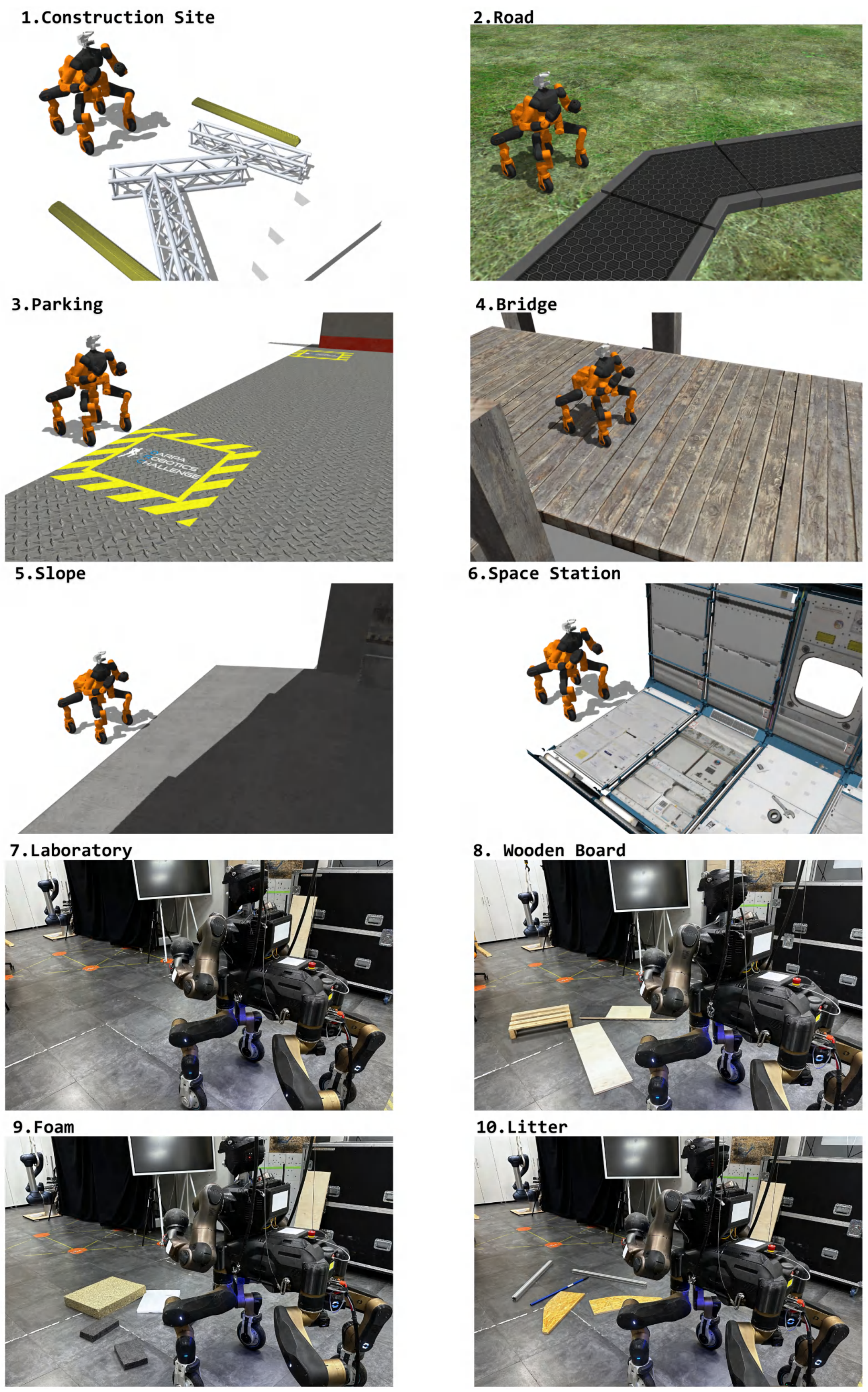


Figure A5: Task scenarios for locomotion morphology selection experiments.

A5.3 VLM Prompts

The prompt words used for the motion morphology selector are shown in the figures, where the prompt words for manipulation morphology selector will be fed into the VLM along with the received textual task instructions from the Behavior Tree.

The motion morphology selector are packaged as one of the functions in ’User Input’ module and it turned ’off’ by default. When it needs to be invoked in task planning, it must be enable in ’Function Options’ or specified to be set to ’on’ when inputting the task instructions.

```
"Suppose you are a humanoid robot and you have two arms, the right arm has a claw gripper
as the end-effector."
"You have two ways to manipulate object: single arm manipulation and dual arm
manipulation."
"You have a camera on your head that can see the object and the environment."
"And you can choose the manipulation method based on the object and task instruction
reasoning."
"When you are doing single arm manipulation, your claw jaw gripper can open up to 10cm."
"You only have the ability to use the jaws to pick up object, regardless of other skillful
grasping methods"
"meaning that you cannot use single arm grasping if the size of the object exceeds the size
of the jaw'opening and closing,"
"or if the object current pose is not capable for grasping with one hand."
"For example: if the given task is 'pick up the drill', and the image shows that a drill is
on the desk."
"Then you will choose to use single arm manipulation, because the size of the drill in the
image can be manipulated with single arm"
"Now you receive the image from the camera and the task below, please answer whether to use
'single arm' or 'dual arm' to do the manipulation."
"(Please answer with only 'single' or 'dual')"
```

Figure A6: Prompts used for Manipulation Morphology Selection.

```
"Suppose you are a robot and you have two ways to move: legs and wheels. "
"You have a depth camera that can obtained the 2D image and point cloud of
the road in front you."

"Now you have to pass the road in front you and here is the 2D image of the
road, and the down sampled point cloud "

"Please choose whether the road should be passed with legs or wheels."
"(Note that wheels are used when the road ahead is flat, or a slope, or the
maximum height of the obstacles is lower than 5 cm."
"And legs should be used when there are obstacles or wooden planks ‘maximum
height higher than 5 cm’. )"
"Determine which type of movement the robot should use to pass through the
roadway. "
"(Please answer with only 'leg' or 'wheel', and the data below is the point
cloud)"
```

Figure A7: Prompts used for Locomotion Morphology Selection.

A6 User Input

The ’User Input’ is the module that links the instructor to the language model and contains predefined prompts for initializing the language system environment and limiting the model output, as well as an interface for accepting task commands sent from the user side.

A6.1 Basic Prompts

Basic prompts provide a description of the task context and robot characteristics, as well as an explanation of user commands and output formatting requirements. As shown below:

```
###Basic Prompts###
"You are now a robot controller, please output a XML file for
constructing a behavior tree to control the robot under the
requirements and given task."
"The robot you control is a centaur like robot, with a humanoid
upper body and four legs, each leg has a wheel at the bottom."
"The robot has two arms, with a claw gripper on the right arm.
It can manipulate objects with two ways: single-arm manipulation
and dual-arm manipulation."
"The robot has two modes of movement: wheel motion and leg motion.
The robot default manipulation and locomotion modes are
'single arm' and 'wheel'."
"The robot has two depth cameras: one located on the head to view
objects, and one on the waist to view the road and terrain ahead."
```

A6.2 Function Options

We designed a number of functions for the robot and packaged them into condition nodes for selective invocation by the LLM during the planning of the task. These functions include: ’Manipulation Morphology Selector’, ’Locomotion Morphology Selector’, ’Failure Detection and Recovery’. We add the descriptions of these functions acting as ’Function Options’ inside the ’User Input’, and set all functions to ’off’ state by default. When the instructor expects a function to be added during a task planning, it can be manually set to ’on’ or include a declaration to use the function in the instruction.

```
###Function Options###
"The robot has the following functions, all of which are 'off' by
    default."
"When a function is 'on', it need to be involved in planning for the
    given task, and when it is 'off', it should not be used."
"Functions: "

"1. 'manipulation_mode_selector': this function allows the robot to
    add the condition node <WhetherSingleArm> to the planning of
    BehaviorTree, which is used to determine whether the current
    manipulation task should use the 'single_arm' or 'dual_arm' type
    of action."

"2. 'locomotion_mode_selector': this function allows the robot to add
    the condition node <WhetherWheelMove> to the planning of the
    behavior tree, which is used to determine whether the current
    locomotion task should use the 'wheel' or 'leg' type of action."

"3. 'detection_recovery': this allows the robot to add the condition
    node <IsActionSuccess>, which is used to determine whether the
    previous action has been successfully completed and, if not, to
    employ a recovery mechanism that repeat the action."
```

A6.3 User Interface

The user interface is responsible for accepting task commands from the instructor and combining them with pre-defined prompt for input to the LLM. The complete user input is as follows.

`User Interface`: hy-motion.github.io/prompt/user_input.ini

`Motion Library`: hy-motion.github.io/prompt/motion_library.ini

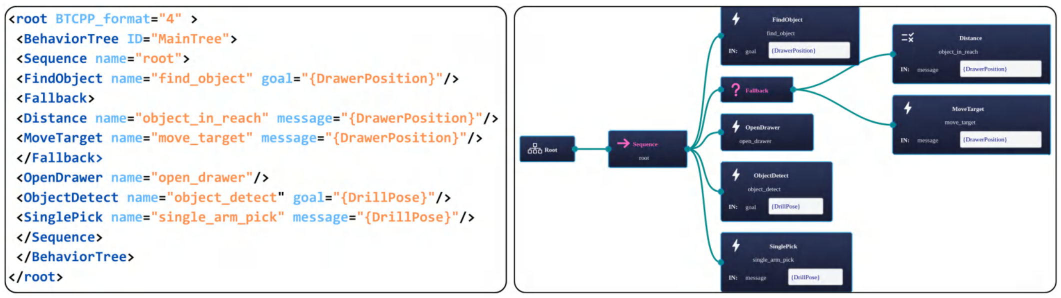
`Basic Prompts`: hy-motion.github.io/prompt/basic_prompt.ini

`Function Options`: hy-motion.github.io/prompt/Function_options.ini

A7 Task Planning with LLM

After receiving the prompts from 'User Input', the LLM output a hierarchical task graph that contains a series of nodes and actions for accomplishing the task. The task graph is saved in an .xml file and serves as a framework for constructing the Behavior Tree that guides the robot's actions. Below we show the detail of experiments in 'Tasks with human instructions' part of Sec. 4.3. For each task, we present the task graph generated by LLM, and the Behavior Tree constructed from it.

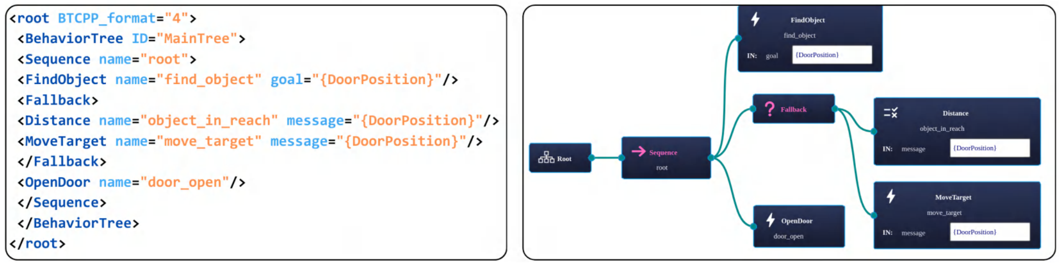
```
Input: Open the drawer and pick up the drill.
```



```
<root BTCPP_format="4" >
 <BehaviorTree ID="MainTree">
 <Sequence name="root">
 <FindObject name="find_object" goal="{DrawerPosition}"/>
 <Fallback>
 <Distance name="object_in_reach" message="{DrawerPosition}"/>
 <MoveTarget name="move_target" message="{DrawerPosition}"/>
 </Fallback>
 <OpenDrawer name="open_drawer"/>
 <ObjectDetect name="object_detect" goal="{DrillPose}"/>
 <SinglePick name="single_arm_pick" message="{DrillPose}"/>
 </Sequence>
 </BehaviorTree>
</root>
```

Figure A8: Task planning of 'Open drawer and pick object'.

```
Input: Find the door and open it.
```



```
<root BTCPP_format="4">
 <BehaviorTree ID="MainTree">
 <Sequence name="root">
 <FindObject name="find_object" goal="{DoorPosition}"/>
 <Fallback>
 <Distance name="object_in_reach" message="{DoorPosition}"/>
 <MoveTarget name="move_target" message="{DoorPosition}"/>
 </Fallback>
 <OpenDoor name="door_open"/>
 </Sequence>
 </BehaviorTree>
</root>
```

Figure A9: Task planning of 'Approach and open door'.

```
Input: Pick up the cracker and put it into the box.
```

```
<root BTCPP_format="4">
<BehaviorTree ID="MainTree">
<Sequence name="root">
<FindObject name="find_object" goal="{CrackerPosition}"/>
<Fallback>
<Distance name="object_in_reach" message="{CrackerPosition}"/>
<MoveTarget name="move_target" message="{CrackerPosition}"/>
</Fallback>
<SinglePick name="single_arm_pick" message="{CrackerPosition}"/>
<FindObject name="find_object" goal="{BoxPosition}"/>
<Fallback>
<Distance name="object_in_reach" message="{BoxPosition}"/>
<MoveTarget name="move_target" message="{BoxPosition}"/>
</Fallback>
<ObjectPlace name="object_place" message="{BoxPosition}"/>
</Sequence>
</BehaviorTree>
</root>
```

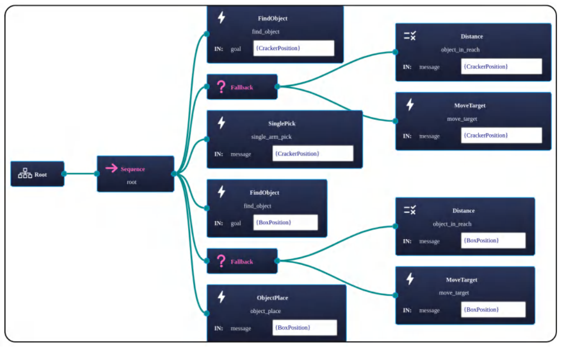


Figure A10: Task planning of 'Pick and place'.

```
Input: Pick up the box and put it on the table.
        ('manipulation_mode_selector'=on)
```

```
<root BTCPP_format="4">
<BehaviorTree ID="MainTree">
<Sequence name="root">
<FindObject name="find_object" goal="{BoxPosition}"/>
<Fallback>
<Distance name="object_in_reach" message="{BoxPosition}"/>
<MoveTarget name="move_target" message="{BoxPosition}"/>
</Fallback>
<ObjectDetect name="object_detect" goal="{BoxPose}"/>
<Fallback>
<Sequence>
<WhetherSingleArm name="whether_single_arm" message="{BoxPose}"/>
<SinglePick name="single_arm_pick" message="{BoxPose}"/>
</Sequence>
<DualPick name="dual_arm_pick" message="{BoxPose}"/>
</Fallback>
<FindObject name="find_object" goal="{TablePosition}"/>
<Fallback>
<Distance name="object_in_reach" message="{TablePosition}"/>
<MoveTarget name="move_target" message="{TableGeneratorPosition}"/>
</Fallback>
<ObjectPlace name="object_place" message="{TablePosition}"/>
</Sequence>
</BehaviorTree>
</root>
```

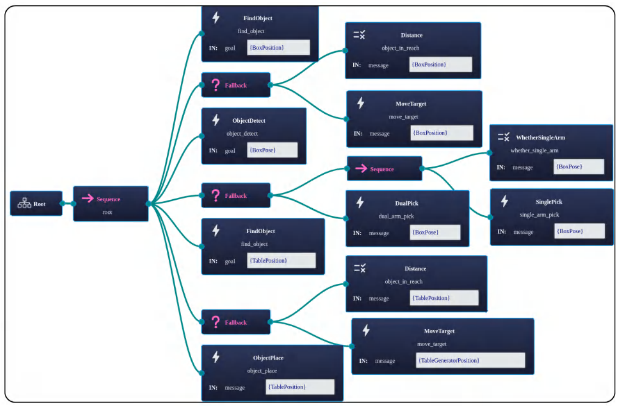


Figure A11: Task planning of 'Dual-arm pick place'.

A8 Long-horizon Task

Environment Setup

The AprilTag system [4], which incorporates a vision-driven algorithm, was used during the long-horizon task to identify the relative objects' location and direction of recognized tags. Within the actual environment, we employ AprilTags to gather task-specific observations. A single visual marker on the door allows for the determination of the door handle's relative position. The robot searches for the tag if it doesn't exit the camera's field of view (FOV). Additionally, AprilTags enable the identification of the drawer's relative positions.

We performed the long-horizon shown in Fig. 1. And the task graph for the long-horizon taks generated by LLM can be found in Fig. 7. For the full video, please refer to https://hy-motion.github.io/

References

[1] A. Laurenzi, D. Antonucci, N. G. Tsagarakis, and L. Muratore. The xbot2 real-time middleware for robotics. *Robotics and Autonomous Systems*, 163:104379, 2023.

[2] A. Laurenzi, E. M. Hoffman, L. Muratore, and N. G. Tsagarakis. Cartesi/o: A ros based real-time capable cartesian control framework. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 591–596. IEEE, 2019.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

[4] E. Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE international conference on robotics and automation*, pages 3400–3407. IEEE, 2011.