LLM $\rightarrow$ Reward Func

LLM to generate the reward function

Hantao Zhou

Universitaet Hamburg

21st December 2023

Overview of Large Language Models in Robotics

- LLMs start combining with robotics.
- Challenges in robotics: hardware dependencies, data lacking.
- New paradigm: LLMs for defining robotic tasks.

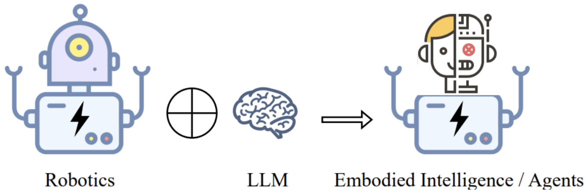


Figure: How are LLM combined with robotics

[5]

Recent Years LLM & Robots

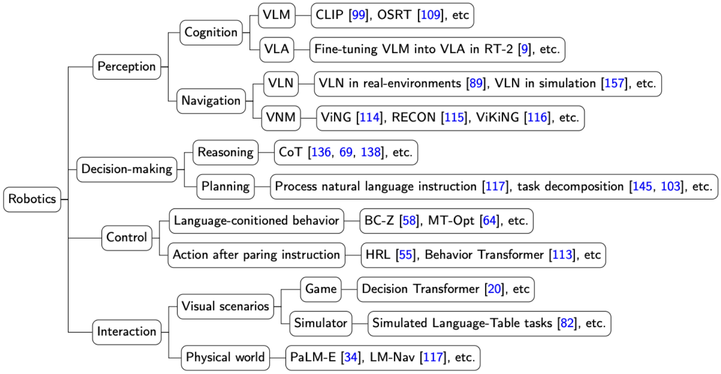

Robotics
Perception
Cognition
VLM
CLIP [99], OSRT [109], etc
VLA
Fine-tuning VLM into VLA in RT-2 [9], etc.
Navigation
VLN
VLN in real-environments [89], VLN in simulation [157], etc.
VNM
ViNG [114], RECON [115], ViKiNG [116], etc.
Decision-making
Reasoning
CoT [136, 69, 138], etc.
Planning
Process natural language instruction [117], task decomposition [145, 103], etc.
Control
Language-conitioned behavior
BC-Z [58], MT-Opt [64], etc.
Action after paring instruction
HRL [55], Behavior Transformer [113], etc
Interaction
Visual scenarios
Game
Decision Transformer [20], etc
Simulator
Simulated Language-Table tasks [82], etc.
Physical world
PaLM-E [34], LM-Nav [117], etc.

Figure: The types of robot-llm tasks

[5]

LLM motion planning histories(1)

1. SayCan Video Link

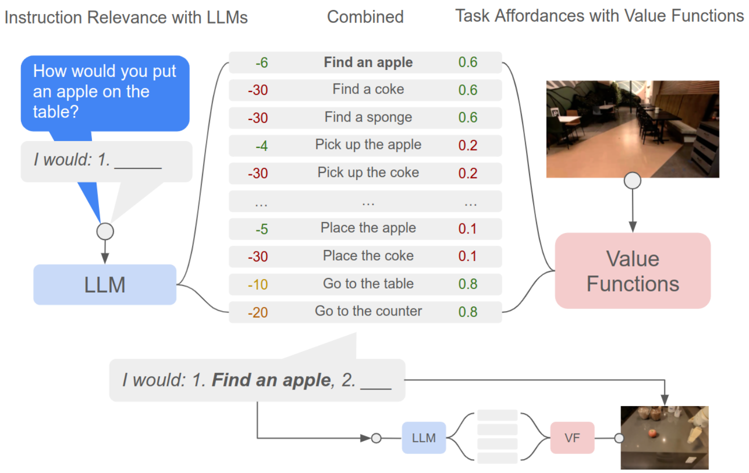
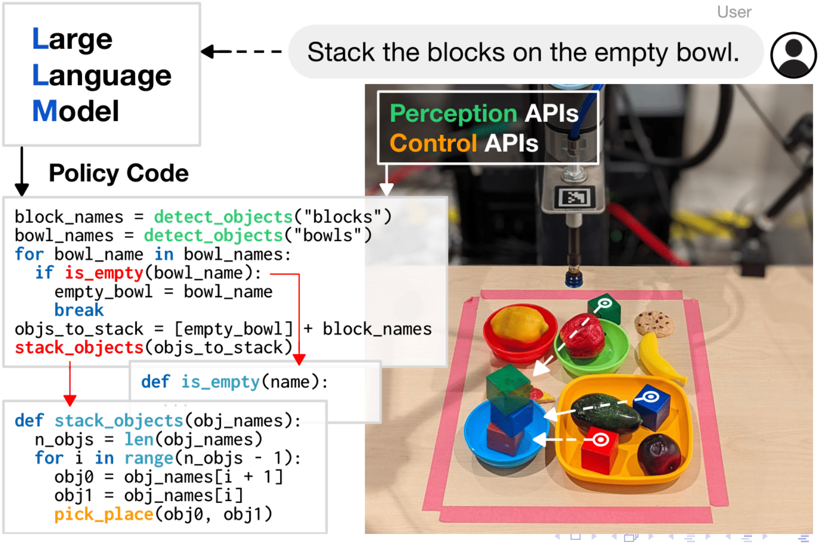


Figure: saycan

LLM motion planning histories(2)

1. Code as Policies Video Link



```
block_names = detect_objects("blocks")
bowl_names = detect_objects("bowls")
for bowl_name in bowl_names:
  if is_empty(bowl_name):
    empty_bowl = bowl_name
    break
objs_to_stack = [empty_bowl] + block_names
stack_objects(objs_to_stack)
```

```
def is_empty(name):
  ...
```

```
def stack_objects(obj_names):
  n_objs = len(obj_names)
  for i in range(n_objs - 1):
    obj0 = obj_names[i + 1]
    obj1 = obj_names[i]
    pick_place(obj0, obj1)
```

Figure: code as policies

Why do we need Reward-based Design

- Previous methods only work on Pick and Place tasks
- Precise motion is also of high importance
- There is no way to code all operations in a hard way

Tasks That Benefit From Reward(1)

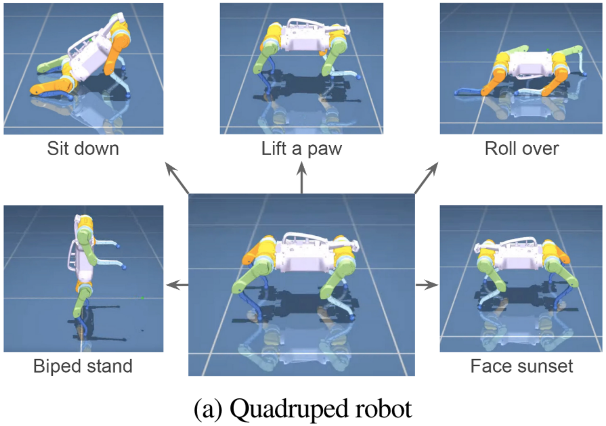


Figure: Quadruped Robotics

[4]

Tasks That Benefit From Reward(2)

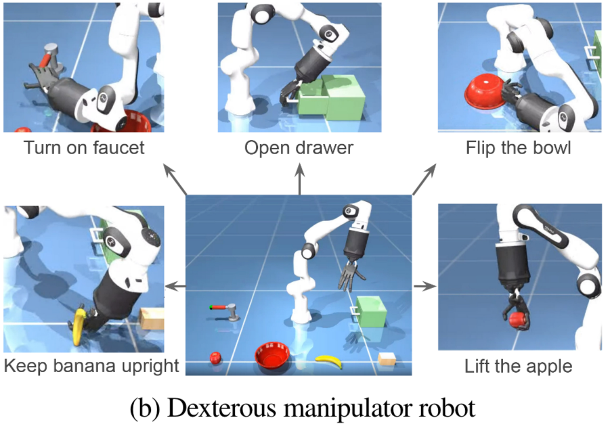


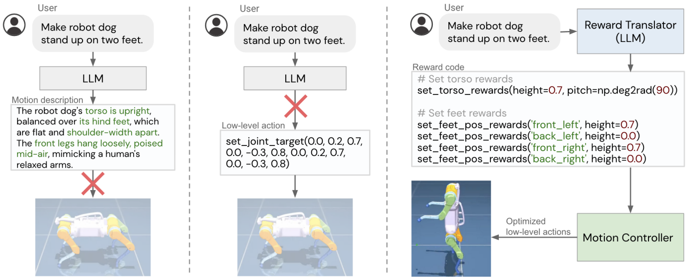
Figure: Shadow Hands

[4]

Language to Rewards

System Overview

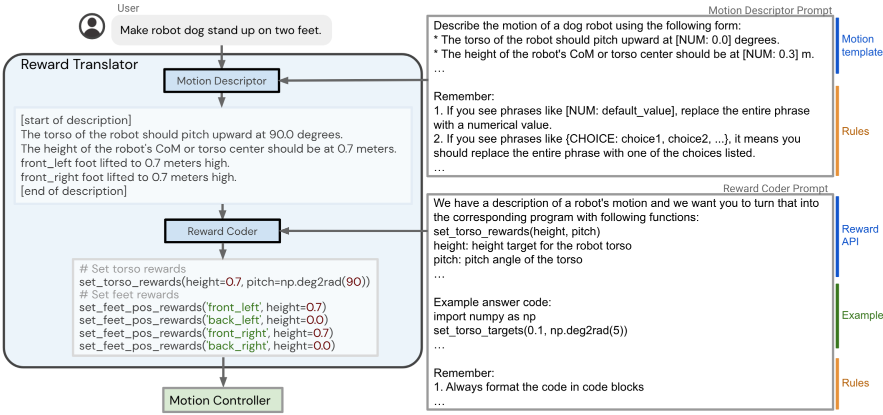
- Reward Translator and Motion Controller.
- Interaction between user inputs, LLMs, and robot actions.


User
Make robot dog stand up on two feet.
LLM
Motion description
The robot dog's torso is upright, balanced over its hind feet, which are flat and shoulder-width apart. The front legs hang loosely, poised mid-air, mimicking a human's relaxed arms.
User
Make robot dog stand up on two feet.
LLM
Low-level action
User
Make robot dog stand up on two feet.
Reward Translator (LLM)
Reward code
Optimized low-level actions
Motion Controller

```
set_joint_target(0.0, 0.2, 0.7,
0.0, -0.3, 0.8, 0.0, 0.2, 0.7,
0.0, -0.3, 0.8)
```

```
# Set torso rewards
set_torso_rewards(height=0.7, pitch=np.deg2rad(90))

# Set feet rewards
set_feet_pos_rewards('front_left', height=0.7)
set_feet_pos_rewards('back_left', height=0.0)
set_feet_pos_rewards('front_right', height=0.7)
set_feet_pos_rewards('back_right', height=0.0)
```

Reward Translator



[start of description]
The torso of the robot should pitch upward at 90.0 degrees.
The height of the robot's CoM or torso center should be at 0.7 meters.
front_left foot lifted to 0.7 meters high.
front_right foot lifted to 0.7 meters high.
[end of description]

```
# Set torso rewards
set_torso_rewards(height=0.7, pitch=np.deg2rad(90))
# Set feet rewards
set_feet_pos_rewards('front_left', height=0.7)
set_feet_pos_rewards('back_left', height=0.0)
set_feet_pos_rewards('front_right', height=0.7)
set_feet_pos_rewards('back_right', height=0.0)
```

Describe the motion of a dog robot using the following form:
* The torso of the robot should pitch upward at [NUM: 0.0] degrees.
* The height of the robot's CoM or torso center should be at [NUM: 0.3] m.
…

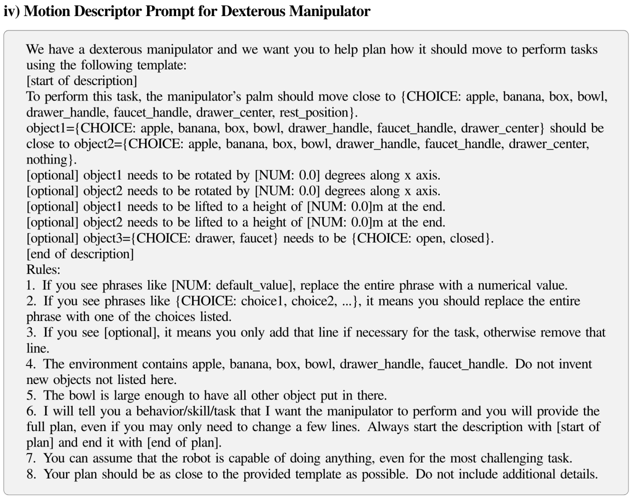
Remember:
1. If you see phrases like [NUM: default_value], replace the entire phrase with a numerical value.
2. If you see phrases like {CHOICE: choice1, choice2, ...}, it means you should replace the entire phrase with one of the choices listed.
…

We have a description of a robot's motion and we want you to turn that into the corresponding program with following functions:
set_torso_rewards(height, pitch)
height: height target for the robot torso
pitch: pitch angle of the torso
…

Example answer code:
import numpy as np
set_torso_targets(0.1, np.deg2rad(5))
…

Remember:
1. Always format the code in code blocks
…

Motion Descriptor



iv) Motion Descriptor Prompt for Dexterous Manipulator

We have a dexterous manipulator and we want you to help plan how it should move to perform tasks using the following template:
[start of description]
To perform this task, the manipulator's palm should move close to {CHOICE: apple, banana, box, bowl, drawer_handle, faucet_handle, drawer_center, rest_position}.
object1={CHOICE: apple, banana, box, bowl, drawer_handle, faucet_handle, drawer_center} should be close to object2={CHOICE: apple, banana, box, bowl, drawer_handle, faucet_handle, drawer_center, nothing}.
[optional] object1 needs to be rotated by [NUM: 0.0] degrees along x axis.
[optional] object2 needs to be rotated by [NUM: 0.0] degrees along x axis.
[optional] object1 needs to be lifted to a height of [NUM: 0.0]m at the end.
[optional] object2 needs to be lifted to a height of [NUM: 0.0]m at the end.
[optional] object3={CHOICE: drawer, faucet} needs to be {CHOICE: open, closed}.
[end of description]
Rules:
1. If you see phrases like [NUM: default_value], replace the entire phrase with a numerical value.
2. If you see phrases like {CHOICE: choice1, choice2, ...}, it means you should replace the entire phrase with one of the choices listed.
3. If you see [optional], it means you only add that line if necessary for the task, otherwise remove that line.
4. The environment contains apple, banana, box, bowl, drawer_handle, faucet_handle. Do not invent new objects not listed here.
5. The bowl is large enough to have all other object put in there.
6. I will tell you a behavior/skill/task that I want the manipulator to perform and you will provide the full plan, even if you may only need to change a few lines. Always start the description with [start of plan] and end it with [end of plan].
7. You can assume that the robot is capable of doing anything, even for the most challenging task.
8. Your plan should be as close to the provided template as possible. Do not include additional details.

Reward Coder

```
def set_joint_fraction_reward(name_joint, fraction)
```

This function sets the joint to a certain value between 0 and 1. 0 means close and 1 means open. name_joint needs to be select from ['drawer', 'faucet']

```
def set_obj_z_position_reward(name_obj, z_height)
```

this term encourages the orientation of name_obj to be close to the height (specified by z_height).

```
def reset_reward()
```

This function resets the reward to default values.
Example plan: To perform this task, the manipulator's palm should move close to object1=apple. object1 should be close to object2=bowl. object2 needs to be rotated by 30 degrees along x axis. object2 needs to be lifted to a height of 1.0.
This is the first plan for a new task.
Example answer code:

```
import numpy as np

reset_reward()
    # This is a new task so reset reward; otherwise we don't need it
set_l2_distance_reward("palm", "apple")
set_l2_distance_reward("apple", "bowl")
set_obj_orientation_reward("bowl", np.deg2rad(30))
set_obj_z_position_reward("bowl", 1.0)

execute_plan(4)
```

Remember:
1. Always format the code in code blocks. In your response execute_plan should be called exactly once at the end.
2. Do not invent new functions or classes. The only allowed functions you can call are the ones listed above. Do not leave unimplemented code blocks in your response.
3. The only allowed library is numpy. Do not import or use any other library.
4. If you are not sure what value to use, just use your best judge. Do not use None for anything.
5. Do not calculate the position or direction of any object (except for the ones provided above). Just use a number directly based on your best guess.
6. You do not need to make the robot do extra things not mentioned in the plan such as stopping the robot.

Results

Sample 1

```
import numpy as np

reset_reward()
    # This is a new task so reset reward; otherwise we don't need it
set_l2_distance_reward("palm", "faucet_handle")
set_obj_orientation_reward("faucet_handle", np.deg2rad(90))
set_joint_fraction_reward("faucet", 1)

execute_plan(2)
```

Sample 2

```
import numpy as np

reset_reward()
    # This is a new task so reset reward; otherwise we don't need it
set_l2_distance_reward("palm", "faucet_handle")
set_joint_fraction_reward("faucet", 1) # 1 means open

execute_plan(2)
```

Reward Coder only

Sample 1

```
import numpy as np

reset_reward()
set_joint_fraction_reward("faucet", 1)

execute_plan()
```

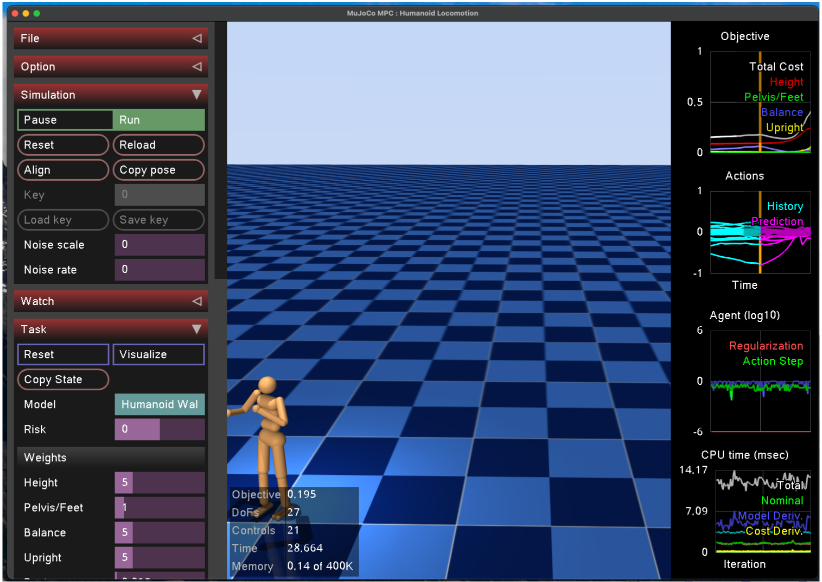
Sample 2

```
import numpy as np

reset_reward() # Reset reward for a new task
set_joint_fraction_reward("faucet",
    1) # Turn on the faucet by setting the joint fraction to 1 (open)

execute_plan(2)
```

Motion Controller

- We have the reward function from the LLMs
- We have the reward computational methods, and what we need exactly is the **behavior**
- The approaches: RL, offline trajectory optimization, receding horizon trajectory optimization(MPC)
    - online methods
    - requires lower computational power

Show MJPC DEMO

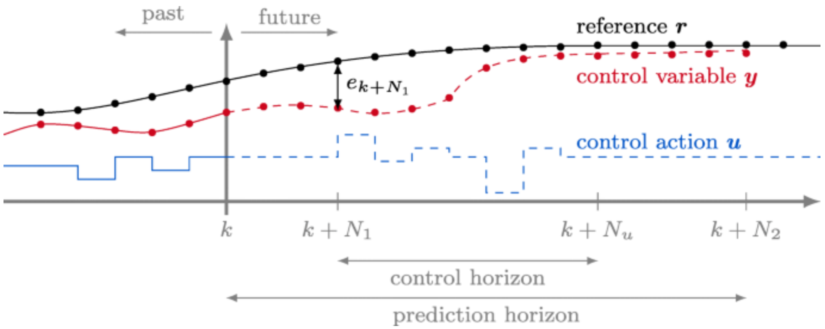

MuJoCo MPC : Humanoid Locomotion

File
Option
Simulation
Pause Run
Reset Reload
Align Copy pose
Key 0
Load key Save key
Noise scale 0
Noise rate 0
Watch
Task
Reset Visualize
Copy State
Model Humanoid Wal
Risk 0
Weights
Height 5
Pelvis/Feet 1
Balance 5
Upright 5

```
Objective  0.195
DoFs       27
Controls   21
Time       28.664
Memory     0.14 of 400K
```

[3]

MPC Overview


past
future
reference $\boldsymbol{r}$
e_{k+N_1}
control variable $\boldsymbol{y}$
control action $\boldsymbol{u}$
k
$k+N_1$
$k+N_u$
$k+N_2$
control horizon
prediction horizon

Basic Steps

1. MPC **plans** a sequence of optimized actions
2. robot carries out the actions
3. advance to the next step and then update the states

Understanding Planners in MJPC(1)

- **Predictive Sampling Planner:**
  - A zero-order, sampling-based predictive control method.
  - Shooting method(Guess where the future will go, simulate and decide, adjust according to observation)
  - Simple, yet competitive with more elaborate derivative-based algorithms.
  - Better at exploring non-smooth optimization landscape
  - Used for the Manipulation task

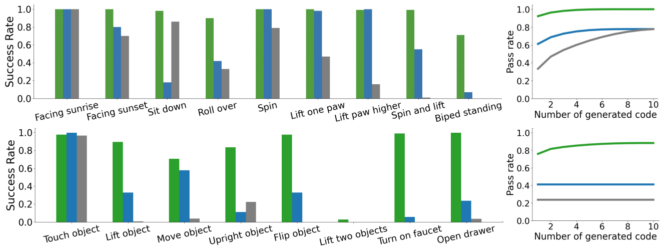
Understanding Planners in MJPC(2)

- **iLQG (Iterative Linear Quadratic Gaussian) Planner:**
    - iteratively updating the control policy based on linear approximations of dynamics and cost
    - Interweaves the local collocations with the shooting mechanisms
    - Produces smoother and more accurate actions
    - Used for legged locomotion tasks

Experimental Targets

- Test Tasks.
  - Quadruped Robotics
  - Dexterous Manipulator
- Goals
  - Sufficient for diverse and complex robot motions?
  - Is the extra motion descriptor effective?
  - Can this method be applied to the real robot?

Evaluation and Results



- Key findings and success rates.
- Ours, Reward Coder Only, Code as Policies

Language to Rewards

Key Concepts

- Transforming language instructions into reward functions.
- Reward functions as flexible task representations.
- **Components:**
  1. Reward Translator (based on LLMs)
  2. Motion Controller (based on MuJoCo MPC)
- Result Video

Takeaways

- Language2Rewards stands as a unique method that applies the reward in a real-time control environment
- The selection of method should be considered case by case
  - MPC / RL / Offline Optimization
  - Different Planners
- Large Language Model can well understand the user's intention and produce the code accordingly in robotics

References I

[1] Michael Ahn et al. *Do As I Can, Not As I Say: Grounding Language in Robotic Affordances*. Aug. 16, 2022. DOI: 10.48550/arXiv.2204.01691. arXiv: 2204.01691 [cs]. URL: http://arxiv.org/abs/2204.01691 (visited on 07/06/2023). preprint.

[2] Jacky Liang et al. *Code as Policies: Language Model Programs for Embodied Control*. May 24, 2023. DOI: 10.48550/arXiv.2209.07753. arXiv: 2209.07753 [cs]. URL: http://arxiv.org/abs/2209.07753 (visited on 12/06/2023). preprint.

References II

[3] Max Schwenzer et al. "Review on Model Predictive Control: An Engineering Perspective". In: *The International Journal of Advanced Manufacturing Technology* 117.5 (Nov. 1, 2021), pp. 1327–1349. ISSN: 1433-3015. DOI: 10.1007/s00170-021-07682-3. URL: https://doi.org/10.1007/s00170-021-07682-3 (visited on 12/20/2023).

[4] Wenhao Yu et al. "Language to Rewards for Robotic Skill Synthesis". In: ().

[5] Fanlong Zeng et al. *Large Language Models for Robotics: A Survey*. Nov. 13, 2023. arXiv: 2311.07226 [cs]. URL: http://arxiv.org/abs/2311.07226 (visited on 11/30/2023). preprint.