Code as Policies:
Language Model Programs for Embodied Control

Presenter: Arpit Bahety

31 October 2023

Motivation

Place the dish a bit to the left

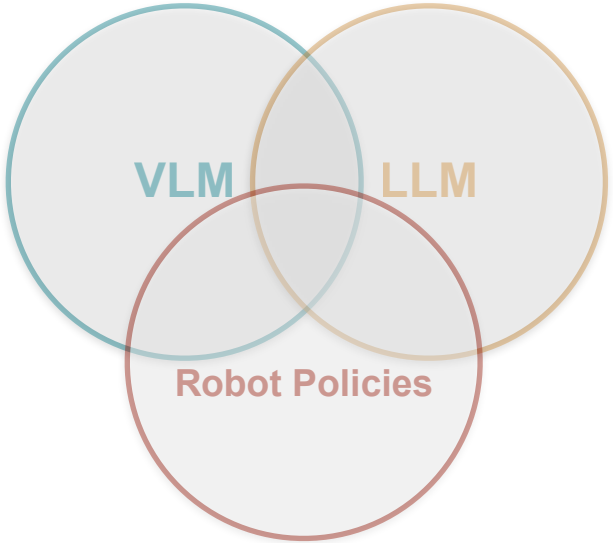
Move the arm slowly near the glass

Put the apple down when you see the plate

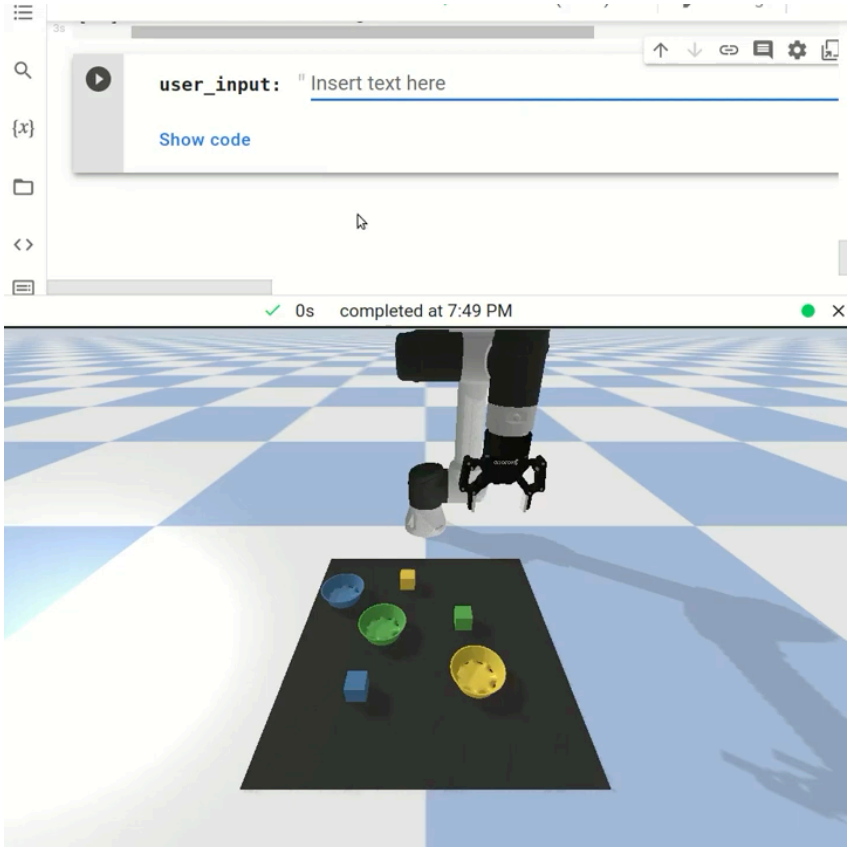
Language in Robotics

Do As I Can, Not As I Say: Grounding Language in Robotic Affordances, CoRL 2022

Language in Robotics



Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language, ICLR 2023



Language in Robotics

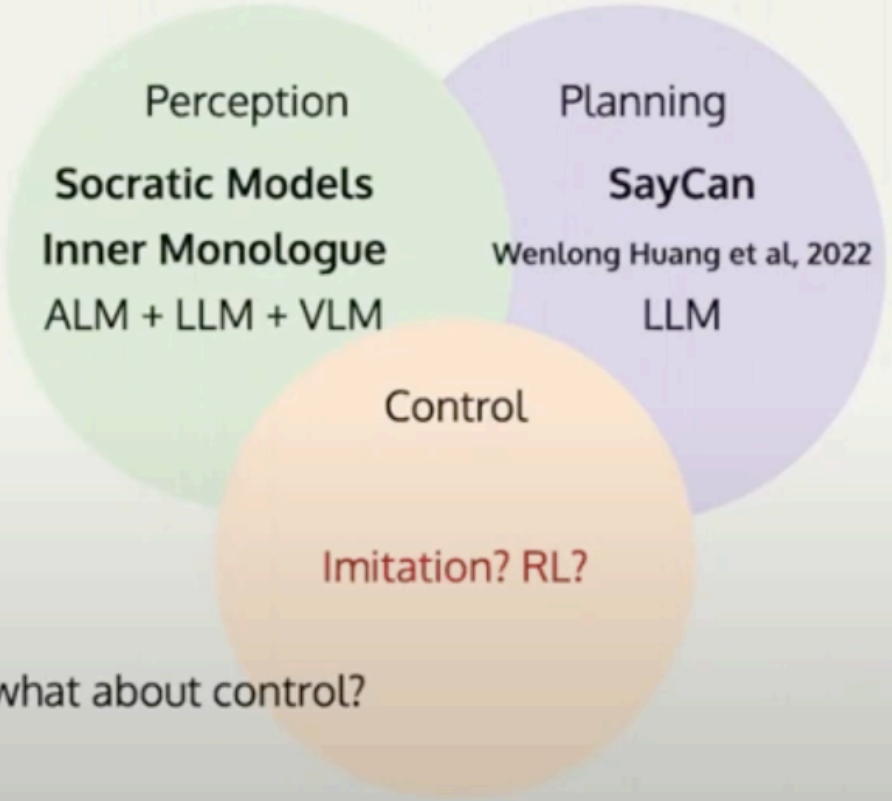
Instruction: Move the coke can a bit to the right

```
LLM Plan [14], [17], [18]
1. Pick up coke can
2. Move a bit right
3. Place coke can
```

```
Socratic Models Plan [16]
objects = [coke can]
1. robot.grasp(coke can)  open vocab
2. robot.place_a_bit_right()
```


[14] Language models as zero-shot planners: Extracting actionable knowledge for embodied agents
[17] Do as i can, not as i say: Grounding language in robotic affordances
[18] Inner monologue: Embodied reasoning through planning with language models
[16] Socratic models: Composing zero-shot multimodal reasoning with language

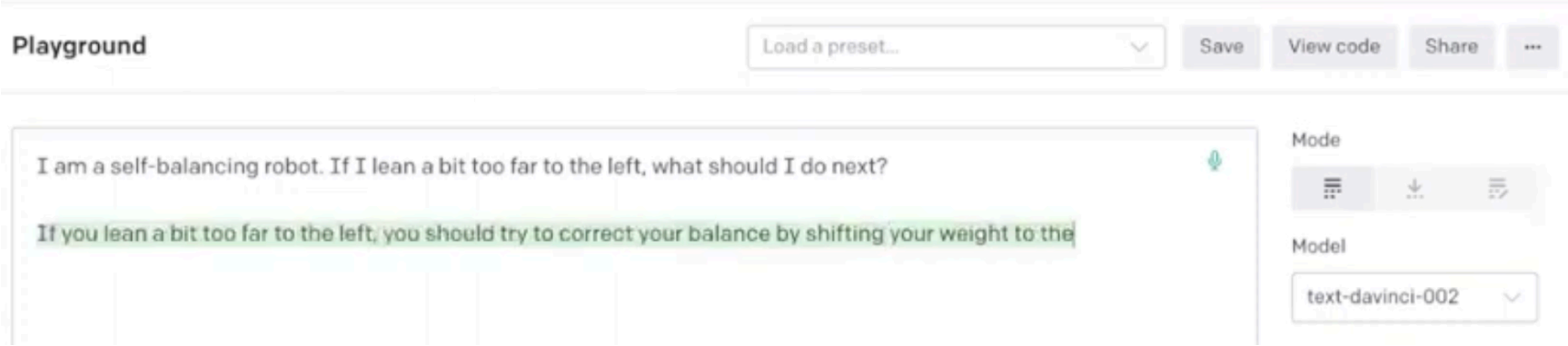
Language in Robotics



- Only for high level? what about control?

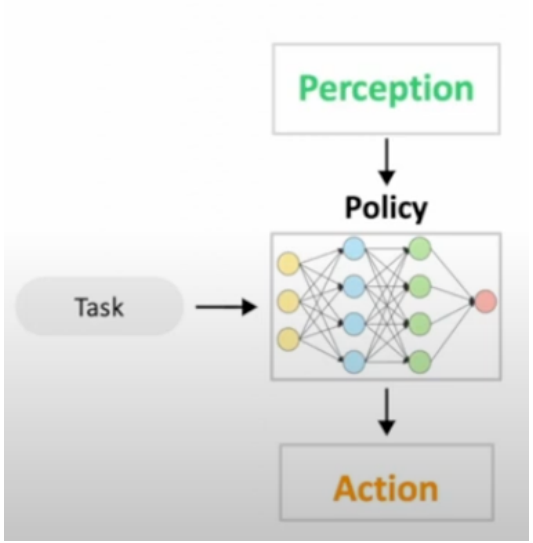
Key insight

Intuition and commonsense is not just a high-level thing but applies to low-level behaviors too!

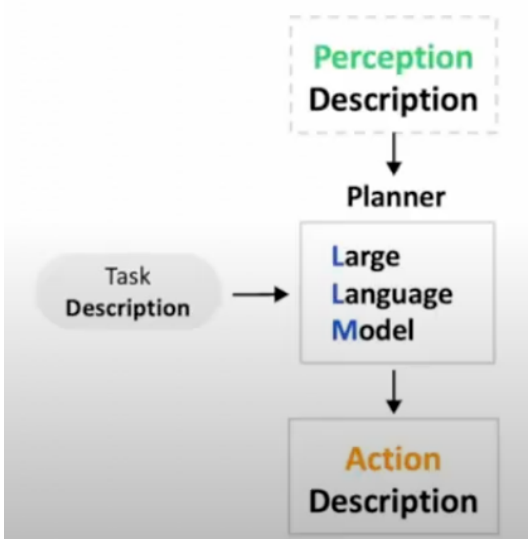


But how to extract it?

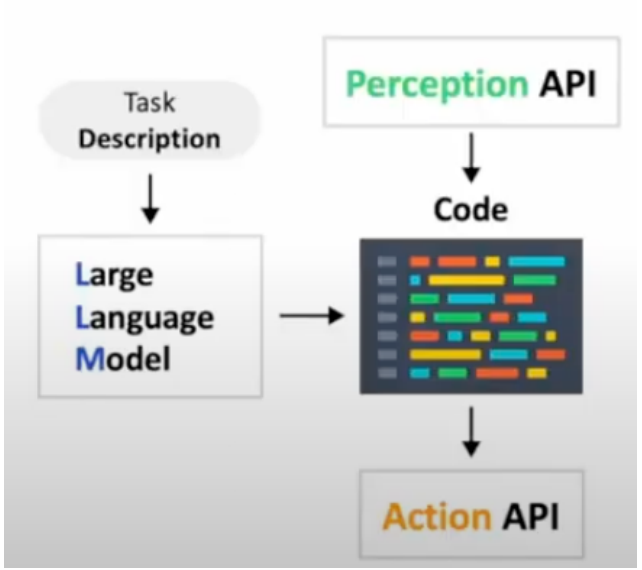
Language Model Programs → Output Code as Policies



Learn Robot Policies



Use LLMs to Plan



Code as Policies
Use LLMs to Write Robot Code

Problem Setting

Large Language Model

User

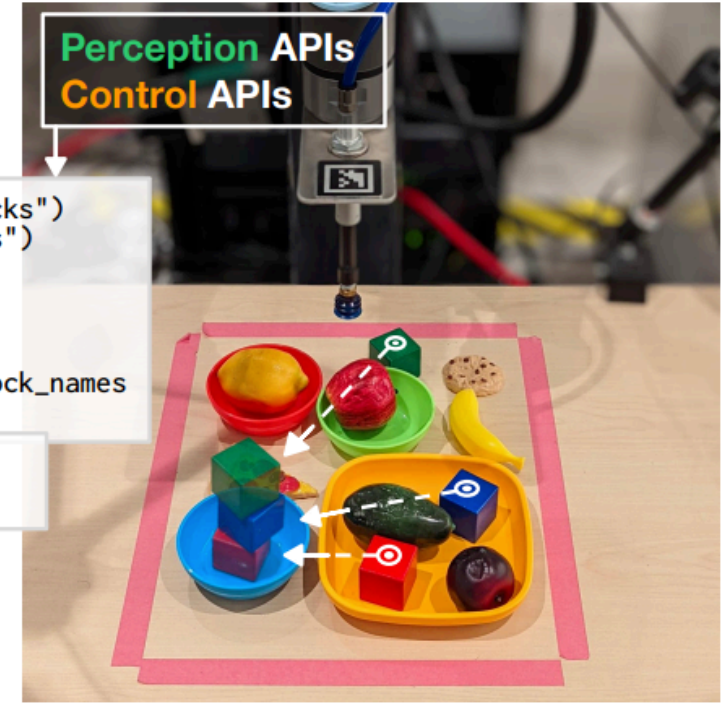
Stack the blocks on the empty bowl.

Policy Code

```
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)
```



And so now..

Instruction: Move the coke can a bit to the right

```
LLM Plan [14], [17], [18]
1. Pick up coke can
2. Move a bit right
3. Place coke can
```

```
Socratic Models Plan [16]
objects = [coke can]
1. robot.grasp(coke can)  open vocab
2. robot.place_a_bit_right()
```

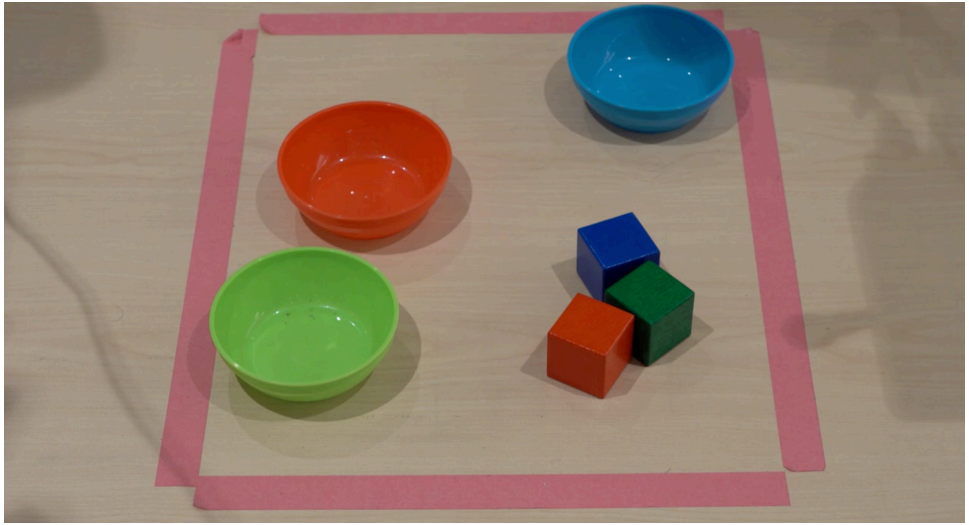
```
Code as Policies (ours)
while not obj_in_gripper("coke can"):
    robot.move_gripper_to("coke can")
robot.close_gripper()
pos = robot.gripper.position
robot.move_gripper(pos.x, pos.y+0.1, pos.z)
robot.open_gripper()
```

Approach

Given examples (via few-shot prompting), map language instruction to code snippets

- React to perceptual inputs
- Parametrize control APIs
- Directly compiled and executed on robot

Put the red block to the left of the rightmost bowl



[PROMPT - High-level UI]

```
objects = ['red block', 'green block', 'blue block', 'red bowl', 'green bowl', 'blue bowl']
# put the red block to the left of the rightmost bowl.
rightmost_bowl_name = parse_obj_name('the rightmost bowl', f'objects = {get_obj_names()}')
if rightmost_bowl_name:
    say(f'Putting the red block to the left of the {rightmost_bowl_name}')
    left_pos = parse_position(f'a point 10cm left of the {rightmost_bowl_name}')
    put_first_on_second('red block', left_pos)
else
    say('There are no bowls')
```

LMPs can be composed

First-party libraries

[PROMPT - Parse Object Names]

[PROMPT - Prase Positions]

```
ock', 'blue block', 'red bowl', 'green bowl', 'blue bowl']
# a point 10cm left of the blue bowl.
blue_bowl_name = parse_obj_name('blue bowl', f'objects = {get_obj_names()}')
blue_bowl_pos = get_obj_pos(blue_bowl_name)mes)
left_obj_pos = blue_bowl_pos + [-0.1, 0]wl_positions)
ret_val = left_obj_pos)wl_names[right_bowl_idx]
ret_val = right_bowl_name
```

LMPs can hierarchically generate functions

```
bowl_positions = get_obj_positions_np(bowl_names)
right_bowl_idx = get_right_most_idx(bowl_positions)
```

[PROMPT - Function Generation]

```
# define function: bowl_positions = get_obj_positions_np(bowl_names).
def get_obj_positions_np(obj_names):
    obj_positions = []
    for obj_name in obj_names:
        obj_positions.append(get_obj_pos(obj_name))
    return np.array(obj_positions)

# define function: right_bowl_idx = get_right_most_idx(bowl_positions).
def get_right_most_idx(points):
    return np.argmax(points[:, 0])
```

Third-party libraries

Experiments

- Domains
  - Code-generation benchmarks
  - Drawing shapes via generated waypoints
  - Real-world table-top manipulation
  - Simulation table-top manipulation
  - Mobile navigation and manipulation

- Hypothesis tested
  - Can LLMs help perform low-level reasoning and control?
  - Evaluate the impact of using hierarchical code generation
  - Analyze modes of generalization

Code-Generation Benchmarks

	GPT-3 [12]		Codex [1]	
Method	6.7B	175B	cushman	davinci
Flat	3	68	54	81
Hierarchical	**5**	**84**	**57**	**95**

HumanEval Python Benchmark

	Greedy	P@1	P@10	P@100
Flat	45.7	34.9	75.1	90.9
Hierarchical	**53.0**	**39.8**	**80.6**	**95.7**

RoboCodeGen Benchmark

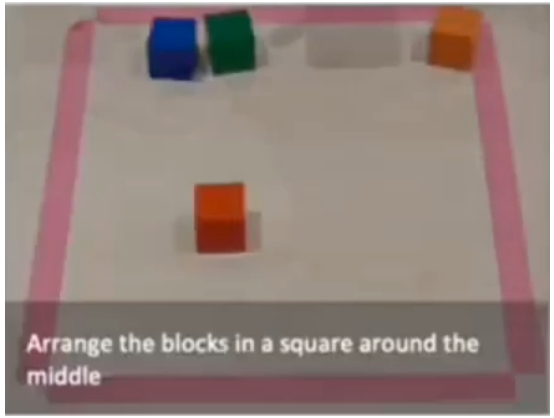
Hierarchical Code-Generation is Better!

Simulation Table Top Manipulation

Code as Policies Leads to Better Generalization

		Frozen LLM	
Task Family	CLIPort	Natural Language	LMP (ours)
Seen Objs. & Seen Insts.			
Long-Horizon	78.80	86.40	**97.20**
Spatial-Geometric	**97.33**	N/A	89.30
Unseen Objs. & Seen Insts.			
Long-Horizon	36.80	88.00	**97.60**
Spatial-Geometric	0.00	N/A	**73.33**
Unseen Objs. & Unseen Insts.			
Long-Horizon	0.00	64.00	**80.00**
Spatial-Geometric	0.01	N/A	**62.00**

Real-World Table Top Manipulation



Mobile Manipulation


```
# This is the compost bin.
compost_bin_pos, compost_bin_angle = get_robot_pos_and_angle()
compost_bin_name = 'compost bin'
say('ok')

# This is the recycle bin.
recycle_bin_pos, recycle_bin_angle = get_robot_pos_and_angle()
recycle_bin_name = 'recycle bin'
say('ok')

# This is the landfill bin.
landfill_bin_pos, landfill_bin_angle = get_robot_pos_and_angle()
landfill_bin_name = 'landfill bin'
say('ok')

# The coke can and the apple are on the table.
coke_can_loc = 'table'
apple_loc = 'table'
say('ok')

# Put way the coke can and the apple on their corresponding bins.
say('got it - putting the coke can and the apple on their corresponding
bins')
goto_loc(coke_can_loc); pick_obj('coke can')
goto_pos(recycle_bin_pos, angle=recycle_bin_angle)
place_at_obj(recycle_bin_name)
goto_loc(apple_loc); pick_obj('apple')
goto_pos(compost_bin_pos, angle=compost_bin_angle)
place_at_obj(compost_bin_name)
```

Discussion of Results

- Can perform low-level reasoning and control
- Hierarchical code generation is useful
- Generalization - interpreting natural language instructions, processing perception outputs, then parameterizing low-dimensional inputs to control primitives

Limitations

- Restricted by Perception APIs:
    - Only support having a set of unique objects
    - Difficult to describe a certain trajectories

- Restricted to the implemented Control APIs
- Limited to examples provided in prompts. e.g. build a house with the blocks.

Future Work and Extended Readings

- Future work for paper
  - Chain-of-thought for code generation
  - Learn control APIs on-the-fly

- Extended Readings
  - Language models as zero-shot planners
  - Say-can
  - Inner-Monologue
  - Socratic Models
  - Cliport

Summary

- Can LLMs help with low-level control and reasoning
- SOTA can generate plans or use pre-defined skills
- Key Idea
  - Write code as policies instead of language
  - Perform hierarchical code-generation
- Zero-shot generalization to various instructions in sim and real-world tasks