

HOMEWORK 10^{*}

AGENTS FOR ROBOTICS[†]

24-880 AI AGENTS FOR ENGINEERS
<https://cmuagents.github.io>

OUT: Sunday, April 5th 2026
DUE: Sunday, April 12th 2026
TAs: Peter Pak
CAs: Hardik Choudhary

Instructions

- **Collaboration Policy:** Please read the collaboration policy in the syllabus.
- **Late Submission Policy:** See the late submission policy in the syllabus.
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code.
 - **Written:** You will submit your completed homework as a PDF to Gradescope. Please use the provided template for the submissions which can be written in $\text{\LaTeX}$ (via [Overleaf](#)) or directly on the PDF document. Each answer should be within the box provided. If you do not follow the template, your assignment may not be graded correctly and there will be a **2% penalty** (e.g., if the homework is out of 100 points, 2 points will be deducted from your final score).
 - **Code:** Gradescope's Autograder will be used to evaluate the coding portions of this assignment. In order to best obtain the points for this assignment, please follow the code upload instructions. If you do not follow the instructions correctly the autograder may not work properly.

Question	Points
SayCan: Grounded Robot Planning (Written)	15
SayCan: Grounded Robot Planning (Programming)	80
Programming Scripts Upload	0
Collaboration and Generative AI Questions	5
Total:	100

^{*}Adapted from CMU 10-623 Generative AI Homework Template

[†]Compiled on Sunday 5th April, 2026 at 21:59

Introduction

In this homework we explore **SayCan** [Ahn et al., 2022], a foundational framework for grounding large language model (LLM) reasoning in robotic affordances. A recurring theme in agentic AI is that a powerful reasoning engine is only useful if its outputs are *executable* in the environment it operates in. SayCan makes this concrete by combining two complementary signals:

1. **Semantic score** (p_{LLM}): how well a candidate skill matches the high-level instruction, as judged by an LLM.
2. **Affordance score** (p_{afford}): the probability that the skill will physically succeed given the robot's current state, estimated by a learned value function (or, in this homework, a rule-based proxy).

The joint score $p_{\text{LLM}} \times p_{\text{afford}}$ is used to greedily select the next skill at each planning step, producing a plan that is both *linguistically sensible* and *physically feasible*.

At a high level, this homework covers:

1. Understanding the SayCan framework and its core challenges.
2. Implementing LLM-based semantic scoring with prompt engineering.
3. Implementing rule-based affordance scoring for a kitchen robot.
4. Combining these into a full SayCan planning loop.

Local Environment

For this homework we'll be using `uv` as the project and package manager. If you haven't installed `uv`, follow the instructions at <https://docs.astral.sh/uv/>.

<code>source .venv/bin/activate</code>	(Linux and Mac) Activates python environment.
<code>source .venv/Scripts/Activate</code>	(Windows) Activates python environment.
<code>uv sync</code>	Installs / synchronizes project dependencies.
<code>uv add <package></code>	Installs a <code>pip</code> package and adds it to dependencies.
<code>uv remove <package></code>	Removes a <code>pip</code> package dependency.
<code>uv run <script.py></code>	Runs a Python script inside the <code>uv</code> environment.

1. Navigate to the project directory and synchronize dependencies:

```
uv sync
```

2. Activate the environment.

Mac and Linux:

```
source .venv/bin/activate
```

Windows:

```
source .venv/Scripts/activate
```

3. Run the starter script to verify your environment:

```
uv run programming/saycan.py
```

1 SayCan: Grounded Robot Planning (Written) (15 points)

SayCan [Ahn et al., 2022] established the paradigm of using an LLM as a high-level task planner while grounding its outputs in the robot's physical capabilities via an affordance model. The framework scores each candidate skill with the joint score:

$$\text{score}(s) = p_{\text{LLM}}(s \mid \text{instruction}) \times p_{\text{afford}}(s \mid \text{robot state})$$

and greedily selects the highest-scoring skill at each planning step.

- 1.1. (5 points) The lecture identifies a *latency mismatch* as one of the four core challenges of combining LLMs with physical robots. Briefly describe the problem and explain how **hierarchical control** addresses it.

- 1.2. (5 points) SayCan uses the joint score $p_{\text{LLM}} \times p_{\text{afford}}$ rather than either term alone. Explain what each term captures and give one concrete example of a situation where relying on **only** p_{LLM} would cause the robot to fail.

- 1.3. (2 points) **Select one:** The lecture identifies four core challenges of merging LLMs with robotics. Which challenge does SayCan's affordance model most directly address?
- ☐ Poor physical world understanding
 - ☐ Latency and real-time control
 - ☐ The sensor-action space mismatch
 - ☐ Safety and robustness
- 1.4. (2 points) **Select one:** In SayCan, what does the affordance model estimate for a candidate skill s given the current robot state?
- ☐ The semantic similarity between the instruction and skill description
 - ☐ The probability that skill s will succeed given the current robot state
 - ☐ The time required to execute skill s
 - ☐ Whether the LLM's instruction is grammatically correct
- 1.5. (1 point) **Select one:** Which of the following is a stated limitation of the SayCan framework?
- ☐ It requires a GPU to run the LLM planner
 - ☐ It cannot handle tasks with more than three steps
 - ☐ All skills must be pre-programmed; the system cannot learn new skills online
 - ☐ It only works in simulation and cannot run on real hardware

2 SayCan: Grounded Robot Planning (Programming) (80 points)

In this section you will implement a SayCan-style planner for a kitchen robot. All of your work goes in `programming/saycan.py`.

Kitchen Setup

The robot operates in a kitchen with the following skill library:

Skill	Description
<code>pick_up_apple</code>	Pick up the apple.
<code>pick_up_mug</code>	Pick up the mug.
<code>pick_up_plate</code>	Pick up the plate.
<code>open_fridge</code>	Open the refrigerator door.
<code>close_fridge</code>	Close the refrigerator door.
<code>place_item_in_fridge</code>	Place the held item inside the refrigerator.
<code>place_item_on_counter</code>	Place the held item on the kitchen counter.
<code>place_item_on_table</code>	Place the held item on the dining table.
<code>fill_mug_with_water</code>	Fill the mug with water from the sink.
<code>open_cabinet</code>	Open the kitchen cabinet.
<code>close_cabinet</code>	Close the kitchen cabinet.
<code>done</code>	The task is complete.

The robot state is a Python dictionary with the following keys:

Key	Type	Description
<code>reachable_objects</code>	<code>list[str]</code>	Objects the robot can currently reach
<code>held_object</code>	<code>str None</code>	Object currently held by the robot
<code>open_containers</code>	<code>list[str]</code>	Containers currently open ("fridge", "cabinet")

The function `execute_skill(skill, robot_state)` is already provided in the starter code — it applies a skill to the current state and returns the updated state. **Do not modify it.**

2.1. (20 points) Implement `llm_score(instruction, skill)`.

This function scores how appropriate `skill` is as the next step by prompting the LLM to output a relevance score. It returns a float in `[0.0, 1.0]`.

1. Build a prompt using `instruction` and `SKILL_DESCRIPTIONS[skill]` that asks the model to respond with a single float from `0.0` (not appropriate) to `1.0` (very appropriate).
2. Call `generate(prompt)` and parse the first float found in the response using a regular expression.
3. Clamp the result to `[0.0, 1.0]` and return it. If no float can be parsed, return `0.0`.

You can test your implementation with:

<code>uv run programming/saycan.py</code>	Runs the end-to-end kitchen demo
<code>uv run test.py saycan</code>	Runs the autograder tests locally

2.2. (15 points) Implement `affordance_score(skill, robot_state)`.

This is a rule-based affordance model that returns 0.9 when the skill is physically feasible and 0.05 when it is not. The skill "done" is a special case and always returns 0.1 (the LLM determines when the task is truly complete).

The feasibility rules for each skill are documented in the starter code docstring.

2.3. (10 points) Implement `joint_score(instruction, skill, robot_state)`.

The joint score is the product of the LLM semantic score and the affordance score:

```
joint_score = llm_score(instruction, skill) × affordance_score(skill, robot_state)
```

2.4. (10 points) Implement `select_skill(instruction, robot_state)`.

Compute the joint score for every skill in `SKILLS` and return the name of the skill with the highest score.

2.5. (25 points) Implement `saycan(instruction, robot_state, max_steps)`.

Run the SayCan planning loop:

1. Select the best skill using `select_skill`.
2. Print the selected skill (for visibility during development).
3. Execute it with `execute_skill` to obtain the updated state.
4. Append the skill name to the plan.
5. If the selected skill is "done", stop the loop.

The loop runs for at most `max_steps` iterations. Return the list of executed skills.

The provided tests are the exact ones that will run on the autograder (i.e. if your local tests pass, they should also pass on the autograder).

3 Programming Scripts Upload (0 points)

To upload your code to Gradescope, run the following command to generate a zip file:

```
uv run programming/zip_programming.py
```

Upload the generated zip file to the appropriate programming slot on Gradescope.

3.1. Make sure the following files are included when uploading to Gradescope:

- ☐ `saycan.py`

4 Collaboration and Generative AI Questions (5 points)

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found in the syllabus.

- 4.1. (2 points) Did you collaborate with anyone on this assignment? If so, list their name or Andrew ID and which problems you worked together on.

- 4.2. (3 points) Did you use generative AI for parts of this assignment? If so, include the input prompts along with links to the conversation.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances, April 2022. URL <https://arxiv.org/abs/2204.01691v2>.