

Foundation Models für Robotik



Abschlussbericht der Kooperationsphase 2024/25 des Kurses KA19

Durchgeführt am Institut für Anthropomatik und Robotik (IAR),
Karlsruher Institut für Technologie (KIT)
Betreut durch Prof. Dr.-Ing. Tamim Asfour, Sebastian Rietsch und Timo Birr

Marika Prolingheuer
marika.prolingheuer@gmx.de

Julius Stahl
julius.stahl@proton.me

Inhaltsverzeichnis

1. Einleitung	2
2. Begriffserklärung	2
3. Material und Methoden	3
3.1. Simulationsumgebung	3
3.2. Robotermodell	4
3.3. API-Befehle	4
3.4. Funktion des LLMs	5
3.5. Interaktion mit dem LLM	6
3.6. Methodik	6
4. Ergebnisse	7
5. Diskussion	8
6. Danksagung	10
A. Quellcode	11
A.1. LiteLLM/main.py	11
A.2. Ausschnitt aus robocasa/main.py	16
A.3. Beispiel-Log	25
A.4. Repository	28
B. Quellen	29

Abstract

The execution of commands given to a robot in natural language is still an open problem in the field of robotics. Recent developments in AI research have yielded promising results, which might be a solution to this problem. Foundation Models, more specifically Large Language Models and Artificial Intelligence assistants, can be used to translate human-given tasks into something the robot can understand by splitting the task into multiple pre-defined subtasks autonomously. The use of a Foundation Model can also theoretically be used to achieve an error detection system by giving the model the current state of the world.

1. Einleitung

Die sichere und flexible Manipulation von physischen Objekten ist wesentlicher Bestandteil der Robotik und stellt ihre zentrale Herausforderung dar. Klassische Steuerungsverfahren basieren auf fest definierten Regeln und festgelegten Szenarien. Dadurch sind sie nur eingeschränkt in der Lage, auf unvorhergesehene Veränderungen in dynamischen Umgebungen zu reagieren. In komplexen Alltagssituationen, wie sie in Privathaushalten oder in der Industrie auftreten, sind hohe Anpassungsfähigkeit und kontextabhängige Entscheidungsfindung gefragt. Deshalb gewinnen in der jüngeren Forschung lernende Systeme und Foundation Models zunehmend an Bedeutung. Insbesondere der Einsatz von Large Language Models (LLMs), wie sie über Dienste wie ChatGPT zugänglich sind, bietet neue Ansätze und Möglichkeiten, um Roboterverhalten flexibel und intuitiv über natürliche Sprache zu steuern.

Aktuelle KI-Assistenten wie ChatGPT sind in der Lage, natürliche Sprachbefehle zu interpretieren und daraus ausführbaren Python-Code zu generieren. Erste wissenschaftliche Arbeiten wie “SayCan”^[1] oder “Code as Policies”^[2] versuchen, LLMs mit Robotersystemen zu verbinden und zeigen, dass LLMs grundsätzlich zur Planung und Steuerung von Aufgaben für Roboter eingesetzt werden können. Trotz dieser Fortschritte sind die Systeme bislang auf stark vereinfachte Annahmen über die Umgebungen beschränkt.

Diese Arbeit untersucht, inwiefern Foundation Models dazu in der Lage sind, die Manipulation einfacher Alltagsobjekte zuverlässig zu steuern. Ziel ist es, die Eingabe von Befehlen an Roboter in natürlicher Sprache zu ermöglichen, wobei die Foundation Models die autonome Zerlegung der Anweisung in kleinere Teilschritte in Form von zielgerichteten Steuerbefehlen übernehmen. Genauer wurde dies anhand einer festgelegten Aufgabe an den Roboter getestet. Es soll Essen in der Mikrowelle aufgewärmt werden.

Dafür ist der folgende Aktionsablauf erforderlich:

- Öffnen der Mikrowelle
- Aufnehmen des Objekts
- Korrektes Platzieren des Objekts in der Mikrowelle
- Schließen der Mikrowelle
- Starten der Mikrowelle durch Knopfdruck

Der Ablauf soll vom LLM generiert und anschließend vom Roboter ausgeführt werden - allein auf Grundlage eines einfachen Prompts in natürlicher Sprache wie “Your objective is to thaw food in a microwave.”

2. Begriffserklärung

Dieser Abschnitt dient der Erklärung von Begriffen, die in dieser Arbeit häufig verwendet werden und deren Kenntnis für ein gutes Verständnis beim Lesen unabdingbar ist.

API und API-Befehle

API ist eine Abkürzung für “Application Programming Interface”, zu deutsch etwa Programmierschnittstelle oder Anwendungsschnittstelle. Die in dieser Arbeit verwendeten “API-Befehle” sind der KI bereitgestellte Funktionen, die diese verwendet, um Anweisungen an den Roboter zu senden. Diese sind für den Roboter die Schnittstelle zur simulierten Welt.

Foundation Models und Large Language Models

Foundation Models sind grundlegend KI-Modelle, welche auf großen Datensätzen trainiert wurden, um in bestimmten Aufgabenbereichen großflächige Fähigkeiten zu erlangen. Beispiele für Foundation Models sind die Large Language Models wie die GPT-Reihe von OpenAI; diese bieten durch bestimmte Trainingsverfahren die Möglichkeit, Texte zu generieren. Durch Finetuning kann das LLM dazu befähigt werden, eine spezifischere Aufgabe auszuführen, wie die Ausgabe von Text in einem bestimmten Format. Dieses Verfahren kommt auch bei Chatbot-Assistenten zum Einsatz, dabei wird das Modell mit speziell dafür ausgelegten und konstruierten Chatverläufen trainiert. Einzelne Nachrichten werden durch Kontrollsequenzen abgegrenzt, die Verwendung und Interpretation dieser wird durch dieses Vorgehen ebenfalls erlernt.

Die oben angesprochene GPT-Reihe von OpenAI hat in den letzten Jahren durch den ebenfalls von OpenAI entwickelten Service ChatGPT an Bekanntheit gewonnen. Diese beinhaltet GPT-3, GPT-4o und GPT-4.1, sowie weitere Modelle. Der Suffix "o" steht dabei für "omni" und weist auf die Multimodalität des Modells hin. So kann GPT-4o Texte, Bilder und Audio als Eingabe verarbeiten. Neben der GPT-Reihe existieren ebenfalls Modelle, die für schrittweises logisches Denken trainiert wurden. Diese Reihe beinhaltet aktuell o1, o3 und o4-mini.

Wrapper

Wrapper sind Funktionen oder Software-Bibliotheken, deren Zweck darin besteht, andere Funktionen auszuführen, oftmals mit angepassten Parametern. Sie dienen der Abstraktion, Simplifizierung und Vereinheitlichung von Code.

Reasoning

Unter Reasoning versteht man im Kontext von Large Language Models die Fähigkeit, Informationen zu verknüpfen, Kausalitäten zu verstehen und mehrschrittige Schlussfolgerungen zu ziehen und dadurch strukturiert zu einer Lösung zu gelangen. Klassische LLMs sind für Sprachgenerierung trainiert und produzieren die wahrscheinlichste Fortsetzung der Eingabe. Reasoning-Modelle legen den Schwerpunkt auf logisches Denken und das Erstellen und Abarbeiten von Instruktionsketten. Dies äußert sich in einer verbesserten Performance im Umgang mit mathematischen Problemen.

3. Material und Methoden

3.1. Simulationsumgebung

Die Experimente wurden in der Simulationsumgebung *MuJoCo*^[3] über Python durchgeführt. *MuJoCo* steht für "Multi-Jointdynamics with Contact" und ist eine physikalisch realistische Simulationsengine, die das Verhalten komplexer Gelenkstrukturen und die Interaktion mit deren Umgebung simuliert. Die *MuJoCo* Codebase ist auf GitHub frei verfügbar.

Als Simulations-Frameworks kamen *robocasa*^[4] und *robosuite*^[5] zum Einsatz. *robosuite* stellt die Robotersteuerung und die Standardumgebung für Manipulationsaufgaben und einen stationären Roboter bereit. *MuJoCo* wird von *robosuite* für die Bewegungen und Kollisionen des Roboters mit Objekten und sich selbst genutzt. *robocasa* erweitert dieses Setup um realitätsnahe, alltägliche Szenarien und bietet die genaue Manipulationsaufgabe sowie eine Erweiterung des Roboters durch eine mobile Basis. Konkret wurde eine vorgefertigte Küchenszene genutzt.

Diese besteht aus

- einer Küchenzeile,
- mehreren interaktiven Objekten, darunter das Objekt, was aufgetaut werden soll und
- einer Mikrowelle mit vertikalem Türgriff und einem Startknopf

Die Szene wird bei jedem Durchlauf zufällig initialisiert. Dabei wird das zu manipulierende Objekt aus einer Liste ausgewählt und auf der Theke platziert.

3.2. Robotermodell



Abb. 1: Das PandaOmron-System mit mobiler Basis in einer *robocasa* Küche

Ein simuliertes *PandaOmron*-System kam als virtuelles Robotermodell zum Einsatz. Dieses verfügt über eine mobile Basis zur freien Bewegung der Plattform und einen Roboterarm, der auf der Plattform montiert ist. Der Endeffektor ist ein Zwei-Finger-Greifer mit Greif- und Loslassfunktion.

3.3. API-Befehle

Die Befehle, die dem Roboter zur Verfügung gestellt wurden, werden “API-Befehle” genannt. Der Roboter soll die Anweisungen, die ihm gegeben werden, in solche API-Befehle zerlegen. Die Befehle wurden selbst entwickelt und in verschiedene Kategorien eingeteilt. Alle Befehle gehen von einem Ursprung des Raums relativ zur Roboterbasis aus und sind auch in Richtung des Roboters orientiert.

Low-Level-Befehle

Im Rahmen der Implementierung wurden sogenannte Low-Level-Befehle entwickelt. Diese lassen sich durch die Nähe zur Hardware charakterisieren, was ebenfalls zu dem Namen führt. Die folgenden Befehle wurden realisiert:

- `get_eef_pos` gibt die Position des Endeffektors an
- `get_eef_rot` gibt die Rotation des Endeffektors an
- `resolve_object_from_name` gibt Position und Rotation eines simulierten Objekts anhand des Namens zurück
- `move_abs` bewegt den Greifer zu den angegebenen absoluten Koordinaten x,y,z

- `open_gripper` öffnet den Greifer
- `close_gripper` schließt den Greifer

High-Level-Befehle

Die High-Level-Befehle basieren auf den Low-Level-Befehlen und kombinieren diese zu hardwareferneren Einheiten, die ganze Aktionen und nicht nur einzelne Bewegungen ausführen können.

- `grip_object_from_above` bewegt den Greifer über das gegebene Objekt und greift es anschließend
- `press_button` drückt den Knopf der Mikrowelle
- `open_door` öffnet die Mikrowellentür
- `close_door` schließt die Mikrowellentür
- `place_object_at_destination` platziert ein Objekt an einer gegebenen Zielposition
- `approach_destination_from_direction` bewegt den Greifer zu den angegebenen absoluten Koordinaten x,y,z . Die Bewegung in die angegebene Richtung wird zuletzt ausgeführt

Wesentlicher Unterschied zwischen den Low-Level-Befehlen und den High-Level-Befehlen ist, dass die High-Level-Befehle symbolische Argumente erwarten. Es werden keine numerischen Werte wie Koordinaten und absolute Rotationen mehr übergeben. Diese werden zu Objekt- und Zielnamen abstrahiert. Viele High-Level-Befehle beinhalten auch komplizierte Bewegungsabläufe, die ohne Argumente ausgeführt werden. Low-Level-Befehle konzentrieren sich auf genau eine Aufgabe.

Kontrollfunktionen

Die Kontrollfunktionen sind Low-Level-Funktionen, die zum Überprüfen des Fortschritts der Aufgabe implementiert wurden. Die einzige solche Funktion ist in unserem Fall `check_gripping_object`, welche anhand des Abstands der Finger überprüft, ob der Greifer aktuell ein Objekt greifen könnte oder ob der Greifer aktuell vollständig geschlossen ist.

3.4. Funktion des LLMs

Large Language Models (LLMs) sind grundlegende KI-Modelle, welche durch das Trainieren auf Texten gelernt haben, wahrscheinlich nachfolgende Buchstabenfolgen vorherzusagen. Damit sind sie in der Lage, menschliche Sprache zu imitieren. Durch Training auf Datensätzen, die logische Probleme und deren Lösungen beinhalten, kann ebenfalls eine Art logisches Denken erzeugt werden. Dieses ist das oben angesprochene Reasoning. Mit weiterem Training kann man diese Modelle auch dazu befähigen, als Chatbot zu fungieren und gegebene Anweisungen zu befolgen. ChatGPT ist ein solcher Chatbot und kann im Dialog einen natürlichen Chatteilnehmer imitieren.

Diese Denkprozesse und die Fähigkeit, Anweisungen zu befolgen, wurden genutzt, um Befehle in natürlicher Sprache durch LLMs zu zerlegen.

3.5. Interaktion mit dem LLM

Für die Interaktion mit dem LLM wurde LiteLLM^[6] verwendet, ein Python-Paket, welches als Wrapper fungiert und Aufrufe an verschiedene LLM-Anbieter unter dem Schema der OpenAI API vereinheitlicht. Ebenso erleichtert LiteLLM die Aufrufe in Python, indem das OpenAI-Schema in parametrisierten Funktionen gebunden wird.

Die OpenAI API ist ein verständlicher und gut dokumentierter Weg, mit Modellen, die OpenAI entwickelt und über diese Schnittstelle bereitstellt, zu kommunizieren. Die Vereinheitlichung der Schemata der verschiedenen Anbieter unter dem Schema der OpenAI API ermöglicht das schnelle Austauschen und somit das Testen von verschiedenen Modellen.

Primär wurden Modelle genutzt, welche von OpenAI entwickelt wurden, darunter o4-mini, GPT-5 und GPT-5 Nano.

3.6. Methodik

Zur Evaluation wurde eine Reihe systematischer Tests durchgeführt. Ziel war es, die Robustheit der Simulation sowie die Leistungsfähigkeit der KI unter verschiedenen Bedingungen zu untersuchen.

Die Simulation und die KI wurden mithilfe folgender Variationen getestet:

1. Variation des KI-Modells
2. Mitgabe eines Bildes, einer Bildbeschreibung in natürlicher Sprache oder im JSON-Format, oder keines der beiden
3. Variation der Benutzung der von OpenAI bereitgestellten Reasoning-Funktion

Die Ausgaben der KI wurden in jedem Durchlauf durch die Ausführung der Simulation verifiziert, dies führte damit zu einem der folgenden vier Ergebnisse:

1. Erfolg, wobei am Ende das Objekt in der Mikrowelle liegt und der Knopf gedrückt ist
2. Fehlschlag, wobei die Aufgabe am Ende nicht vollständig erreicht wurde
3. Error, wobei die KI einen Befehl falsch aufruft und der Ablauf unterbrochen wird
4. Timeout, wobei die Simulation nach fünf Minuten nicht beendet ist, was ein Anzeichen für ein Hängenbleiben des Roboterarms ist

Insgesamt wurden 22 Variationen simuliert. Jede Variation wurde in 10 Durchläufen getestet, um statistische Aussagen zu ermöglichen. Ein Simulationsdurchlauf lief folgendermaßen ab:

1. Dem Sprachmodell wurde die Aufgabe gegeben, das Objekt in die Mikrowelle zu legen. Die Weitergabe von Bilddaten hing von der Konfiguration der Simulation ab.
2. Das Sprachmodell generierte einen Plan, wie die Aufgabe mit den API-Befehlen zu bewältigen ist.
3. Der Plan wurde schrittweise von der KI ausgeführt, indem diese entsprechende API-Befehle ausführte und mögliche Rückgaben dieser übermittelt bekam
4. Der Erfolg der KI wurde anhand des Simulationszustandes in einer der obigen Kategorien eingeteilt

4. Ergebnisse

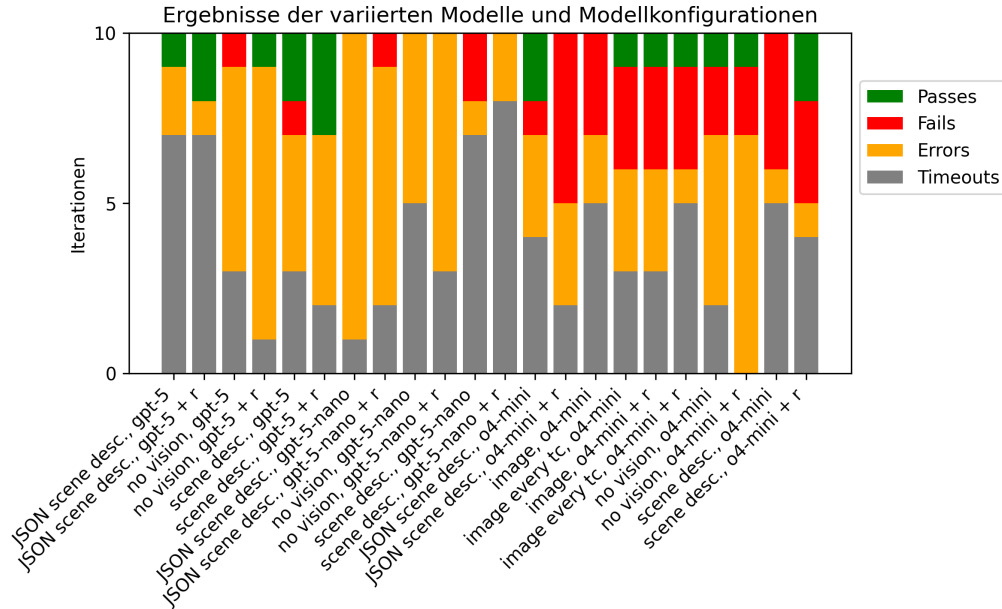


Abb. 2: Ergebnisse der variierten Modelle und Modellkonfigurationen

Abb. 2 zeigt die Resultate der untersuchten Modelle und Modellkonfigurationen. Hierbei steht

- “no vision” für keine Verarbeitung von Bilddaten
- “scene desc.” für das Anfügen einer KI-gefertigten Szenenbeschreibung des gleichen Modells
- “JSON scene desc.” für das Anfügen einer KI-gefertigten Szenenbeschreibung im JSON-Format
- “image” für das Senden eines Bilds ohne Szenenbeschreibung und
- “image every tc” für das automatische Übergeben eines Bildes nach jedem Tool-Call.

Der Suffix “+ r” kennzeichnet die Verwendung von reasoning. Die Bezeichnungen o4-mini und GPT-5 sowie GPT-5 nano sind die Modelle, die in den Durchläufen verwendet wurden.

- Insgesamt waren 18 von 220 Iterationen erfolgreich.
- In 12 von 22 Simulationsvariationen wurden Erfolge erzielt.
- Die Kombination des Modells GPT-5 mit textueller Szenenbeschreibung und Reasoning erreichte mit 3 Passes die meisten erfolgreichen Simulationsdurchläufe.
- Die Hauptgründe für einen Fehlschlag der Simulation waren Errors (43%) und Timeouts (41%). Fehler durch Fails traten mit 17% am seltensten auf.
- In allen manuell evaluierten Simulationen, die zu Fails führten, war die Ursache ein vertauschter Ablaufplan des LLMs: Der Plan sah fälschlicherweise vor, die Mikrowelle erst nach dem Greifen des Objekts zu öffnen, was den Fehler verursachte.

- Mit dem Modell GPT-5 nano wurden keine Passes erzielt. Mit dem Modell o4-mini liefen 9% der Simulationen durch, mit GPT-5 waren es 15%.
- Das Modell o4-mini ohne Szenenbeschreibung und Bilddaten mit Reasoning endete als einziges der Kombinationen in keiner Iteration in einem Timeout.

Tabelle 1: Erfolgsraten der Modellkonfigurationen

Modellkonfiguration	Durchgelaufene Iterationen	Erfolgsquote	Erfolgsquote ohne GPT-5 nano
scene desc.	7 von 60	11,66%	17,50%
no vision	3 von 60	5%	7,50%
JSON scene desc.	5 von 60	8,33%	12,50%
image	3 von 40	7,50%	-
image every tc	2 von 20	10%	-
reasoning	11 von 110	10%	13,75%
Gesamt	18 von 220	8,11%	11,25%

Tabelle 1 zeigt die Ergebnisse in Abhängigkeit von den Modellkonfigurationen.

Tabelle 2: Ergebnisse der Modelle und Modellkonfigurationen

Model	gpt-5						gpt-5-nano						o4-mini									
Vision Type	JSON		none		text		JSON		none		text		JSON		img		img+tc		none		text	
Reasoning	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-
Passes	2	1	1	0	3	2	0	0	0	0	0	0	0	2	1	0	1	1	1	1	2	0
Fails	0	0	0	1	0	1	1	0	0	0	0	2	5	1	3	3	3	3	2	2	3	4
Errors	1	2	8	6	5	4	7	9	7	5	2	1	3	3	3	2	1	3	7	5	1	1
Timeouts	7	7	1	3	2	3	2	1	3	5	8	7	2	4	3	5	5	3	0	2	4	5

Tabelle 2 zeigt die Anzahl der Ergebnisse aller Modellkonfigurationen, differenziert nach Passes, Fails, Errors und Timeouts.

5. Diskussion

Die Analyse der Simulationsergebnisse zeigt eine auffällige Diskrepanz zwischen den Erfolgsquoten einzelner Modell- und Konfigurationskombinationen (55%) und der Gesamtperformance (8%) über alle

Durchläufe. Die Differenz zeigt, dass einige Modelle in der Lage sind, die Aufgabe erfolgreich zu lösen. Alle Variationen funktionieren jedoch nur sehr inkonsistent oder gar nicht. Diese Ergebnisse könnten darauf hinweisen, dass die KI zwar Potenzial besitzt, aber in unserer Implementierung nicht robust genug ist, um die Aufgabe zuverlässig in allen Umgebungen zu erfüllen.

Ebenso auffällig sind die hohen Anteile an Errors und Timeouts. Diese könnten auf Schwierigkeiten der KI bei der Prozesssteuerung oder Fehler in unserer API hindeuten. Die Timeouts können auch im Hängenbleiben des Roboters an Küchenmobiliar begründet sein. Die Simulationsparameter beinhalteten hohe Reibungskoeffizienten, um das Fallenlassen von kugelförmigen Lebensmitteln zu vermeiden. Das Hängenbleiben könnte dadurch begünstigt worden sein.

Das Modell GPT-5 nano erzielte keinen erfolgreichen Simulationsdurchlauf. Dieses Modell hat eine wesentlich geringere logische Denkfähigkeit als GPT-5 und o4-mini. Im Gegenzug ist seine Geschwindigkeit höher und die Kosten pro Million Tokens mit 0.05\$ im Vergleich zu den anderen mit 1.25\$ und 1.10\$ geringer. Es lässt sich vermuten, dass für eine erfolgreiche Objektmanipulation in komplexen Umgebungen eine höhere Denkfähigkeit des LLMs erforderlich ist.

Mit einer Erfolgsquote von 15% performte GPT-5 besser als o4-mini (9%). Dieses Ergebnis legt nahe, dass die "kleineren" Modelle (GPT-5 nano und o4-mini) nicht robust genug für komplexe Steuerungsaufgaben sind.

Ein Vergleich der Konfigurationen zeigt deutliche Unterschiede in der Erfolgsquote. Da alle Simulationen mit GPT-5 nano fehlschlagen, lassen sich diese Ergebnisse nicht den getesteten Konfigurationen, sondern dem Modell selbst zuschreiben. Entsprechend können sie keine belastbare Aussage über den Konfigurationserfolg liefern und werden nicht in die Auswertung einbezogen.

Die Konfiguration mit textueller Szenenbeschreibung zeigt einen positiven Effekt auf die Erfolgsquote. Mit 17,5% Erfolgen unter allen Durchläufen mit GPT-5 und o4-mini ist diese um mehr als 6 Prozentpunkte höher als die durchschnittliche Quote. Eine Szenenbeschreibung in natürlicher Sprache scheint für die Planungsfähigkeit der Modelle vorteilhaft zu sein.

Die JSON-basierten Szenenbeschreibungen führten dagegen zu keinem erheblichen Effekt auf die Erfolgsraten. Dies deutet darauf hin, dass die zusätzliche Formalisierung in JSON keinen Vorteil gegenüber den natürlichsprachlichen Beschreibungen bringt. Die Konfiguration ohne Bilddaten und Szenenbeschreibung erzielte mit 7,5% die niedrigsten Erfolgsraten. Ohne visuelle Informationen scheint es für die KI deutlich schwieriger, die Aufgabe zuverlässig zu bewältigen.

Die Analyse der Reasoning-Funktion zeigt einen leichten Vorteil (13,75) gegenüber dem Durchschnitt (11,25). Bei genauerer Untersuchung wird klar, dass der Vorteil expliziten Reasonings vom KI-Modell abhängig ist. Bei GPT-5 nano führte auch die zusätzliche Reasoning-Fähigkeit zu keinem erfolgreichen Durchlauf. Mit o4-mini liefen 5 von 60 Iterationen (8,33%) durch, was ebenfalls keine Verbesserung zur eigenen Erfolgsrate von 9% darstellt. GPT-5 erzielte mit zusätzlichem Reasoning eine Erfolgsquote von 20%. Das ist eine Verbesserung von einem Drittel zur eigenen Gesamterfolgsquote. Es liegt nahe, dass Reasoning für große, leistungsstarke Modelle wie GPT-5 einen Vorteil bietet. Für kleine Modelle scheint es eher den gegenteiligen Effekt zu haben.

Obwohl einige Fragen beantwortet werden konnten, bleiben offene Aspekte, zu denen weiter geforscht werden sollte. Ein zentraler Ansatzpunkt ist dabei die stärkere Variation sowohl der Aufgabenstellung als auch der verwendeten Prompts. Zukünftige Arbeiten erfordern außerdem eine Erweiterung des Projektumfangs. Aus Kostengründen beschränkten wir uns für diese Arbeit auf 10 Durchläufe pro Variation. Die Ergebnisse sind deshalb statistisch nur begrenzt aussagekräftig und zeigen eher Tendenzen als Fakten auf. Da KI-Ausgaben nicht deterministisch sind, sind Hunderte Durchläufe nötig, um verlässliche Aussagen zu treffen.

6. Danksagung

Unser großer Dank gilt Prof. Dr.-Ing. Tamim Asfour, der das Projekt angeboten und begleitet hat, und Sebastian Rietsch und Timo Birr, die uns in das Thema eingeführt haben und uns stets mit Erklärungen und Tipps zur Seite standen, wenn wir Fragen hatten.

Wir danken unseren Kursleitern, die uns über die Jahre begleitet haben, insbesondere Herrn Gruber, der uns über die Dauer dieses Projekts unterstützt hat.

Zu guter Letzt gilt unser Dank Dr. Hans-Werner Hector und Josefine Hector, die mit ihrer Stiftung das Hector-Seminar finanzieren und uns damit die vielfältigen Erfahrungen der letzten Jahre ermöglicht haben.

A. Quellcode

A.1. LiteLLM/main.py

```

1  from os import environ
2  from typing import List
3  import argparse
4  from pathlib import Path
5
6  import litellm
7  import json
8  import logging
9  from sys import stdout
10
11 from Code.LiteLLM.utils import high_level_control_functions
12 from Code.LiteLLM.image_logger import ImageLogger
13 from Code.robocasa_env.main import Controller
14 from Code.LiteLLM.scene_description import get_scene_description,
   ↪ get_scene_description_json
15
16 cur_dir = Path(__file__).parent
17 logs_dir = cur_dir / "Logs"
18
19 environ["OPENAI_API_KEY"] = open(cur_dir / "API_KEY", mode="r").read()
20
21 parser = argparse.ArgumentParser(
22     prog='AutoBatchRobocasaLLM',
23     description='Executes the given batch with the current configuration')
24
25 parser.add_argument('-b', '--batch-name', type=str, default='')
26 parser.add_argument('-v', '--vision-enabled', action='store_true')
27 parser.add_argument('--vision-legacy', action='store_true')
28 parser.add_argument('-m', '--model', default='o4-mini')
29 parser.add_argument('-r', '--renderer', action='store_true')
30 parser.add_argument('-p', '--log-pictures', action='store_true')
31 parser.add_argument('-j', '--use-json', action='store_true')
32 parser.add_argument('-R', '--use-reasoning', action='store_true')
33 parser.add_argument('-s', '--send-picture-every-tool-call', action='store_true')
34
35 args = parser.parse_args()
36
37 batch_name = args.batch_name # leave empty to save in Logs directly
38 batch_path = logs_dir / batch_name
39 batch_path.mkdir(parents=True, exist_ok=True)
40 number_of_logs = sum(
41     1 for f in batch_path.iterdir()
42     if f.is_dir() and f.name.startswith("RobocasaLLM_")
43 )
44 log_path = batch_path / f"RobocasaLLM_{number_of_logs}"
45 log_path.mkdir(parents=True)
46
47 # file handler
48 file_handler = logging.FileHandler(log_path / 'RobocasaLLM.log', mode='w')

```

```

49
50 # console handler
51 console_handler = logging.StreamHandler(stdout)
52
53 # create logger
54 logger = logging.getLogger("RobocasaLLM")
55 logger.setLevel("INFO")
56 logger.addHandler(file_handler)
57 logger.addHandler(console_handler)
58 logger.propagate = False
59
60 vision_legacy = args.vision_legacy
61 vision_enabled = not vision_legacy and args.vision_enabled
62 vision_send_picture_every_tool_call = vision_legacy and args.send_picture_every_tool_call
63 headless = not args.renderer
64
65 with open(log_path / "args.json", mode="w") as args_file:
66     json.dump(vars(args), args_file)
67
68 controller = Controller(headless=headless)
69 controller.start()
70
71 image_logger = ImageLogger(controller, log_path)
72
73 if vision_enabled:
74     if args.use_json:
75         scene_description = get_scene_description_json(image_logger, args.model)
76     else:
77         scene_description = get_scene_description(image_logger, args.model)
78     system_prompt = {"role": "system",
79                     "content": "You may only use one function call per response and have
80                               ↳ to wait for "
81                               "it to finish that you can potentially react to errors
82                               ↳ that arise "
83                               "during execution and are returned by the function. If
84                               ↳ multiple "
85                               "function calls are provided, only the first one will be
86                               ↳ executed. "
87                               "You will get a scene description from the user, given
88                               ↳ this description, "
89                               "think of a plan on how to achieve the task and send a
90                               ↳ message "
91                               "containing the plan. Afterwards, begin with the
92                               ↳ execution."
93                     }
94
95 user_prompt = {"role": "user", "content": [
96     {
97         "type": "text",
98         "text": f"Here's a scene description{' in JSON format' if args.use_json else
99               ↳ '':\n {scene_description}\n"
100         "You are the one-armed robot with a single gripper. Your objective is
101         ↳ to thaw food in a microwave. "

```

```

93         "The object is called \"obj\" in the simulation, the microwave is
94         ↪ called \"container\". "
95         "In the end, the food should be in the microwave, the microwave should
96         ↪ be turned on "
97         "and you should be at least 25 cm away from the object. "
98         "After formulating your plan, immediately begin execution."
99     }
100 ]
101 }
102 else:
103     # You may only use one function call per response and have to wait for it to finish
104     ↪ that you can potentially react
105     # to errors that arise during execution and are returned by the function.
106     # Before attempting to call a function, send one message reasoning which step would be
107     ↪ useful to achieve your goal.
108     # Afterward, execute the next logical step.
109     system_prompt = {"role": "system", "content":
110         "You may only use one function call per response and have to wait for
111         ↪ "
112         "it to finish that you can potentially react to errors that arise "
113         "during execution and are returned by the function. If multiple "
114         "function calls are provided, only the first one will be executed. "
115         f"First,{ ' describe the image provided, then' if vision_legacy else
116         ↪ '' } "
117         "think of a plan on how to achieve the task and send a message "
118         f"containing the {'image description and ' if vision_legacy else ''}"
119         "plan. Afterwards, begin with the execution."
120     }
121
122     # The microwave door is closed.
123     # You should regularly check if the object didn't fall down, as that may happen often.
124     user_prompt = {"role": "user", "content": [
125         {
126             "type": "text",
127             "text":
128                 "You are a one-armed robot with a single gripper. Your objective is to
129                 ↪ thaw food in a microwave. "
130                 "The object is called \"obj\" in the simulation, the microwave is
131                 ↪ called \"container\". "
132                 "In the end, the food should be in the microwave, the microwave should
133                 ↪ be turned on "
134                 "and you should be at least 25 cm away from the object."
135                 f"{' The microwave door is closed.' if not vision_legacy and not
136                 ↪ vision_enabled else ''}"
137         }
138     ]
139 }
140
141 messages: List = [system_prompt, user_prompt]
142 if vision_legacy:
143     image_logger.add_current_scene_to_message(messages[1])
144
145 logger.info(f"Using system prompt:\n{system_prompt['content']}\n")
146 logger.info(f"Using microwave prompt:\n{messages[1]['content'][0]['text']}")

```

```

136 available_functions, tools = high_level_control_functions(controller)
137 assert len(available_functions) == len(tools)
138
139 activate_tools = False
140 used_tool_calls = []
141
142 error_state = False
143 try:
144     while not controller.check_successful() and len(messages) <= 12: # fixed limit of ten
145         ↪ messages + sys + user
146         response = litellm.completion(
147             model=args.model,
148             messages=messages,
149             tools=tools if activate_tools else None,
150             reasoning_effort="medium" if args.use_reasoning and not activate_tools else
151                 ↪ None
152         )
153         # response.usage contains tokens
154         logger.info(f"\nLLM Response:\n{response.choices[0].message.content}")
155         response_message = response.choices[0].message
156         tool_calls = response_message.tool_calls
157
158         # Note: the JSON response may not always be valid; be sure to handle errors
159         messages.append(response_message) # extend conversation with assistant's reply
160
161         # Step 2: check if the model wanted to call a function
162         if tool_calls:
163             logger.info("LLM wants to execute tool calls")
164             logger.info("\nTool calls:")
165             for n, tool_call in enumerate(tool_calls[:]): # create a copy of tool_calls
166                 name = tool_call.function.name
167                 f_args = json.loads(tool_call.function.arguments)
168                 color = "\033[36m"
169                 reset = "\033[0m"
170                 logger.info(f"{color + 'Will not be executed: ' if n > 0 else ''}"
171                             f"{name}({', '.join([f'{arg}={val}' for arg, val in
172                                     ↪ f_args.items()])}){reset if n > 0 else ''}")
173
174                 if n > 0:
175                     tool_calls.pop() # pop one element for each element after the first
176                     ↪ one
177
178         # Step 3: call the function
179
180         # Step 4: send the info for each function call and function response to the
181         ↪ model
182         tool_call = tool_calls[0]
183         function_name = tool_call.function.name
184         function_to_call = available_functions[function_name]
185         function_args = json.loads(tool_call.function.arguments)
186         function_response = function_to_call(**function_args)
187         messages.append(
188             {

```

```

184         "tool_call_id": tool_call.id,
185         "role": "tool",
186         "name": function_name,
187         "content": str(function_response),
188     }
189     ) # extend conversation with function response
190
191     used_tool_calls.append(tool_call)
192 else:
193     logger.info("LLM is reasoning")
194     logger.info(f"\nLLM Reasoning:\n{response.choices[0].message.content}")
195     activate_tools = True
196 if vision_legacy or vision_enabled or args.log_pictures:
197     image = image_logger.get_image()
198     if vision_send_picture_every_tool_call:
199         message = {"role": "user", "content": [
200             {
201                 "type": "text",
202                 "text": "This is the current scene, you may continue the task
203                     ↳ according to the situation after "
204                     "checking if everything is correct."}]
205             }
206         image_logger.add_image_to_message(message, image)
207         messages.append(message)
208 except Exception as e:
209     logger.error(f"Execution failed and yielded following error:\n{e}")
210     logger.info("ERROR")
211     error_state = True
212 finally:
213     if error_state:
214         pass
215     elif controller.check_successful():
216         logger.info("Task accomplished successfully!")
217         logger.info("SUCCESS")
218     else:
219         logger.info("Task failed after ten messages...\n"
220             "Current State:\n"
221             f"Object inside the microwave:
222             ↳ {controller.check_object_in_microwave()}\n"
223             f"Microwave button was pressed: {controller.check_button_pressed()}\n"
224             f"Gripper is at least 25cm away from the door:
225             ↳ {controller.check_gripper_away_from_microwave()}\n")
226         logger.info("Manually check if the procedure was correct:")
227         for tool_call in used_tool_calls:
228             name = tool_call.function.name
229             args = json.loads(tool_call.function.arguments)
230             logger.info(f"{name}({', '.join([f'{arg}={val}' for arg, val in
231                 ↳ args.items()])})")
232         logger.info("FAIL")
233 controller.stop()

```

A.2. Ausschnitt aus robocasa/main.py

```

1  from math import atan2, pi
2  from time import sleep
3  import threading
4
5  from typing import Sequence
6
7  import numpy as np
8  from scipy.spatial.transform import Rotation
9
10 import mujoco
11
12 import robosuite as suite
13 from robosuite.controllers.composite.composite_controller_factory import
14     ↪ load_composite_controller_config
15
16 # noinspection PyUnresolvedReferences
17 import robocasa # needed for the environments, doesn't find them otherwise
18 from robocasa.utils.object_utils import compute_rel_transform, obj_inside_of,
19     ↪ gripper_obj_far
20
21 class Controller:
22     def __init__(self, headless=False):
23         self.headless = headless
24
25         self.max_velocity = 0.3
26         self.min_velocity = 0.03
27
28         self.max_angle_velocity = 0.1
29         self.min_angle_velocity = 0.01
30
31         options = {
32             "env_name": "MicrowaveThawing",
33             "robots": "PandaOmron",
34             "controller_configs": load_composite_controller_config(robot="PandaOmron"),
35             "layout_ids": [0], # change for different kitchen layout
36             "style_ids": [0] # change for different kitchen style
37         }
38         self.env = suite.make(
39             **options,
40             has_renderer=not headless,
41             has_offscreen_renderer=True,
42             render_camera=None,
43             ignore_done=True,
44             use_camera_obs=False,
45             control_freq=10,
46             renderer="mjviewer",
47             camera_names="robot0_agentview_center",
48             camera_heights=720, # Height in pixels
49             camera_widths=1280 # Width in pixels

```

```

50     self.env.reset()
51
52     self.action_dim = self.env.action_spec[0].shape[0]
53
54     # array that contains all movement velocities that are to be executed in the next
55     ↪ timestep
56     # consists of xyz-velocities [0:3], xyz-rotation [3:6], gripper [6] and unknown
57     ↪ [7:12]
58     self.movement = np.zeros(self.action_dim)
59
60     self.simulation_is_running = False
61
62     self.simulation = threading.Thread(target=self._simulate)
63
64     if not headless:
65         self.env.viewer.set_camera(camera_id=2)
66
67     self.env.objects["obj"].friction = (10, 3, 1)
68
69     self.ran_successfully = False
70
71     def _simulate(self) -> None:
72         while self.simulation_is_running and not self.check_successful():
73             self.env.step(self.movement)
74             if not self.headless:
75                 self.env.render()
76             # when the simulation is finished
77             self.ran_successfully = self.check_successful()
78             self.simulation_is_running = False
79             if not self.headless:
80                 self.env.close_renderer()
81             self.env.reset()
82
83     def start(self) -> None:
84         self.simulation_is_running = True
85         self.ran_successfully = False
86         self.simulation.start()
87
88     def stop(self) -> None:
89         self.simulation_is_running = False
90         self.simulation.join()
91
92     def get_vision_data(self):
93         return np.flipud(self.env.sim.render(camera_name="robot0_agentview_center",
94             height=720, # Height in pixels
95             width=1280, # Width in pixels
96             depth=False
97             ))
98
99     def check_gripping_object(self) -> bool:
100         return sum(abs(self.env.observation_spec()["robot0_gripper_qpos"])) > 0.0011

```

```

101 def check_object_in_microwave(self):
102     return obj_inside_of(self.env, "obj", self.env.microwave)
103
104 def check_button_pressed(self):
105     return self.env.microwave.get_state()["turned_on"]
106
107 def check_gripper_away_from_microwave(self):
108     return gripper_obj_far(self.env)
109
110 def check_successful(self):
111     return self.check_object_in_microwave() \
112            and self.check_button_pressed() \
113            and self.check_gripper_away_from_microwave() \
114            or self.ran_successfully
115
116 def transform_to_robot_frame(self, coordinates: Sequence[int],
↪ orientation=np.identity(3)) \
117     -> (np.ndarray, np.ndarray):
118     """transforms coordinates and orientations as rotation matrices into the robot
↪ frame"""
119     return compute_rel_transform(self.env.robots[0].base_pos,
↪ self.env.robots[0].base_ori, coordinates, orientation)
120
121 def get_eef_pos(self) -> np.ndarray:
122     """returns the eef pos relative to the robot frame"""
123     return self.transform_to_robot_frame(self.env.observation_spec()["robot0_eef_pos"],
↪ )][0]
124
125 def get_eef_rot(self) -> np.ndarray:
126     """returns the eef rotation relative to the robot frame"""
127     return Rotation.from_matrix(
128         self.transform_to_robot_frame(
129             self.env.observation_spec()["robot0_eef_pos"],
130             Rotation.from_quat(self.env.observation_spec()["robot0_eef_quat"]).as_mat_
↪ rix()[1]
131         ).as_euler("xyz") / pi * 180
132
133 def resolve_object_from_name(self, object_name: str) -> dict[str, list]:
134     """finds position and rotation of the object with the given name and returns it in
↪ the robot frame"""
135     observation = self.env.observation_spec()
136     print(f'Resolved "{object_name}" '
137           f'to ("pos": {observation[f"{object_name}_pos"]} "quat":
↪ {observation[f"{object_name}_quat"]}'))
138     result = self.transform_to_robot_frame(
139         observation[f"{object_name}_pos"],
140         Rotation.from_quat(observation[f"{object_name}_quat"]).as_matrix()
141     )
142     return {"pos": result[0], "rot": result[1]}
143
144 def open_gripper(self) -> None:
145     """opens gripper"""
146     self.movement[6] = -1

```

```

147     while (self.env.observation_spec()["robot0_gripper_qpos"][0] < 0.0395 and
148            self.env.observation_spec()["robot0_gripper_qpos"][1] > -0.0395):
149         pass
150     self.movement[6] = 0
151     print("opened gripper")
152
153 def close_gripper(self) -> None:
154     """closes gripper"""
155     self.movement[6] = 1
156     sleep(0.1)
157     while np.max(abs(self.env.observation_spec()["robot0_gripper_qvel"])) > 0.01:
158         pass
159     self.movement[6] = 0
160     print("closed gripper")
161     # return self.check_gripping_object()
162
163 def move_abs(self, x, y, z) -> bool:
164     """move to given coordinates relative to the robot frame"""
165     print(f"Started moving to relative coordinates {x, y, z}")
166     prior_vector = None
167     prior_timestep = self.env.timestep
168     while np.max( # compute the maximum difference of start and goal
169                 abs(np.array([x, y, z]) - self.get_eef_pos())
170             ) > 0.02:
171         vector = np.array([x, y, z]) - self.get_eef_pos()
172         distance = np.linalg.norm(vector)
173         velocity = 0
174         if distance > 0.02:
175             velocity = self.max_velocity
176         elif distance > 0.002:
177             velocity = self.min_velocity
178         velocities = vector / distance * velocity
179         self.movement[:3] = velocities
180         # if self.env.timestep % 80 == 0:
181         #     if prior_vector is not None and sum(abs(prior_vector - vector)) < 0.1 \
182         #        and self.env.timestep != prior_timestep:
183         #         return False
184         #     else:
185         #         prior_vector = vector
186         #         prior_timestep = self.env.timestep
187         # if self.env.timestep % 500 == 0:
188         #     print(f"Currently at relative coordinates {self.get_eef_pos()}")
189     self.movement[:3] = np.zeros(3)
190     print(f"moved to relative coordinates {x}, {y}, {z}")
191     return True
192
193 def rotate_gripper_abs(self, end_rotation: Sequence[int]) -> None:
194     """Rotates the gripper to an absolute end rotation relative to the robot frame"""
195     # if end_rotation is quaternion, convert it to euler
196     if len(end_rotation) == 4:
197         end_rotation = quat_to_euler(end_rotation)
198     # if it is not yet euler, it was neither quaternion nor euler, throw error
199     if len(end_rotation) != 3:

```

```

200         raise ValueError(f'"values" must either be euler rotation (3 values) '
201                           f'or quaternion (4 values), not {len(end_rotation)} values')
202     # while the angle differences are bigger than the tolerance
203     while np.max(abs(subtract_angles(end_rotation, self.get_eef_rot()))) > 1:
204         vector = subtract_angles(end_rotation, self.get_eef_rot())
205         angle_velocities = []
206         for rotation in vector:
207             angle_velocity = 0
208             if abs(rotation) > 5:
209                 angle_velocity = self.max_angle_velocity
210             elif abs(rotation) > 1:
211                 angle_velocity = self.min_angle_velocity
212             if rotation < 0:
213                 angle_velocity = -angle_velocity
214             angle_velocities.append(angle_velocity)
215
216         self.movement[3:6] = angle_velocities
217
218         # break if the angle difference is nearly zero for all three axes
219         if all(v == 0 for v in angle_velocities):
220             break
221
222         self.movement[3:6] = np.zeros(3)
223         print(f'rotated gripper to {"", ".join([str(rot) for rot in end_rotation])}')
224
225     def rotate_axis(self, end_rotation: Sequence[int], axis: int) -> None:
226         """Rotates around one axis using only quaternions"""
227
228         if len(end_rotation) == 3:
229             end_rotation = Rotation.from_euler('xyz', end_rotation,
230                                               ↪ degrees=True).as_quat(False)
231         elif len(end_rotation) != 4:
232             raise ValueError(f'"values" must be either Euler rotation (3 values) '
233                               f'or quaternion (4 values), not {len(end_rotation)} values')
234
235         while True:
236             current_quat = self.env.observation_spec()["robot0_eef_quat"]
237             # current_quat = Rotation.from_matrix(np.dot(Rotation.from_quat(self.env.obse_
238             ↪ rvation_spec()["robot0_eef_quat"]).as_matrix(),
239             ↪ self.env.robots[0].base_ori)).as_quat(False)
240
241             # calculate the rotation required to get from the current rotation to the
242             ↪ target rotation and convert it to a vector
243             rotation_diff = Rotation.from_quat(end_rotation) *
244             ↪ Rotation.from_quat(current_quat).inv()
245             rotvec = rotation_diff.as_rotvec()
246
247             # break if the distance to the target rotation is less than 0.1
248             if np.linalg.norm(rotvec) < 0.1:
249                 break
250
251             # Set rotation_speed to a value between max and min speed, matching the
252             ↪ rotation-distance-vector

```

```

247     rotation_speed = np.clip(rotvec, -self.max_angle_velocity,
    ↪     self.max_angle_velocity)
248     if abs(rotation_speed[axis]) < self.min_angle_velocity:
249         break
250
251     axis_vector = np.zeros(3)
252     axis_vector[axis] = rotation_speed[axis]
253
254     self.movement[3:6] = axis_vector
255
256     self.movement[3:6] = np.zeros(3)
257     print("done")
258
259 def approach_destination_from_direction(self, destination: Sequence[float], direction:
    ↪ str) -> bool:
260     """move along every axis except for the given direction, then move to destination
261     uses coordinates relative to the robot frame"""
262     match direction:
263         case "front":
264             matrix = np.array([
265                 [0, 0, 0],
266                 [0, 1, 0],
267                 [0, 0, 1]
268             ])
269         case "left":
270             matrix = np.array([
271                 [1, 0, 0],
272                 [0, 0, 0],
273                 [0, 0, 1]
274             ])
275         case "right":
276             matrix = np.array([
277                 [1, 0, 0],
278                 [0, 0, 0],
279                 [0, 0, 1]
280             ])
281         case "up":
282             matrix = np.array([
283                 [1, 0, 0],
284                 [0, 1, 0],
285                 [0, 0, 0]
286             ])
287         case _:
288             matrix = np.identity(3)
289     if not self.move_abs(*(
290         np.dot(matrix, destination)
291         + np.dot(np.identity(3, dtype=int) - matrix, self.get_eef_pos())
292     )) or not self.move_abs(*destination):
293         return False
294     return True
295
296
297 def grip_object_from_above(self, object_name: str) -> bool:

```

```

298     """Opens gripper, moves gripper to the object with the given name, then closes
299     ↪ gripper"""
300     self.open_gripper()
301     self.move_abs(*(self.resolve_object_from_name(object_name)["pos"] + [0, 0, 0.15]))
302     self.rotate_gripper_abs([180, 0, 0])
303     self.move_abs(*(self.resolve_object_from_name(object_name)["pos"] + [0, 0,
304     ↪ -0.01]))
305     self.close_gripper()
306     print(f'picked object "{object_name}"')
307     return True
308
309 def press_button(self) -> bool:
310     """presses the button of the microwave"""
311     button_pos_abs = self.env.sim.data.get_body_xpos(self.env.microwave.door_name) \
312     + np.dot(np.array([-0.22, -0.33, -0.105]),
313     ↪ self.env.robots[0].base_ori.T)
314     button_pos_rel = self.transform_to_robot_frame(button_pos_abs)[0]
315     self.rotate_axis([-180, -90, 0], 1)
316     self.close_gripper()
317     if not self.move_abs(*(self.get_eef_pos() + [-0.1, 0, 0])):
318         return False
319     self.approach_destination_from_direction(button_pos_rel, "front")
320     self.movement[0] = self.max_velocity
321     sleep(1)
322     self.movement[0] = 0
323     return True
324
325 def open_door(self) -> bool:
326     """opens the microwave door"""
327     # alternatively self.env.sim.data.get_body_xpos(self.env.microwave.door_name)
328     handle_pos_abs = self.env.microwave.pos + np.dot(np.array([-0.22, -0.18, 0]),
329     ↪ self.env.robots[0].base_ori.T)
330     # self.render_coordinate_frame(*handle_pos_abs, None)
331     # controller.env.microwave.pos - controller.env.microwave.size*[-0.21, 0.5, 0]
332     handle_pos_rel = self.transform_to_robot_frame(handle_pos_abs)[0]
333     self.open_gripper()
334     if not self.move_abs(*(self.get_eef_pos() + [-0.1, 0, 0])):
335         return False
336     self.rotate_axis([-180, -90, 0], 1)
337     self.approach_destination_from_direction(handle_pos_rel, "front")
338     self.close_gripper()
339     joint_pos, _ = self.transform_to_robot_frame(self.env.sim.data.joint(self.env.microwave.joints[0]).xanchor)
340     ↪ vector = (self.get_eef_pos() - joint_pos)[:2]
341     vector = vector / np.linalg.norm(vector)
342     while abs(vector[1]) > 0.15:
343         tangential_vector = np.dot(vector, np.array([[0, -1], [1, 0]]))
344         self.movement[0:2] = tangential_vector * self.max_velocity
345
346         vector = (self.get_eef_pos() - joint_pos)[:2]
347         vector = vector / np.linalg.norm(vector)
348     self.movement[0:2] = np.zeros(2)
349     self.open_gripper()

```

```

346         if not (self.move_abs(*(self.get_eef_pos() + [-0.1, 0, 0]))
347                 and self.move_abs(*(self.get_eef_pos() + [0, -0.25, 0]))
348                 and self.move_abs(*(self.get_eef_pos() + [0.1, 0, 0]))):
349             return False
350         vector = (self.get_eef_pos() - joint_pos)[:2]
351         vector = vector / np.linalg.norm(vector)
352         # possibly in world coordinates, requires testing
353         self.movement[1] = self.max_velocity
354         while abs(vector[1]) > 0.2:
355             vector = (self.get_eef_pos() - joint_pos)[:2]
356             vector = vector / np.linalg.norm(vector)
357         self.movement[1] = 0
358         if not self.move_abs(*(self.get_eef_pos() + [0.15, -0.1, -0.26])):
359             return False
360         print("opened door")
361         return True
362
363     def close_door(self) -> bool:
364         """closes the microwave door"""
365         joint_pos, _ = self.transform_to_robot_frame(self.env.sim.data.joint(self.env.microw
366             ↪ rowave.joints[0]).xanchor)
367         microwave_pos = self.transform_to_robot_frame(self.env.microwave.pos)[0]
368         if not (self.move_abs(*(microwave_pos + [-0.4, 0.1, -0.05]))
369                 and self.move_abs(*(microwave_pos + [-0.3, 0.1, -0.3]))):
370             return False
371         self.rotate_axis([-180, -90, 0], 1)
372         self.approach_destination_from_direction(
373             [joint_pos[0] - 0.15, joint_pos[1] + 0.2, microwave_pos[2] - 0.08], "up"
374         )
375         if not self.move_abs(joint_pos[0] - 0.15, microwave_pos[1], microwave_pos[2] -
376             ↪ 0.05):
377             return False
378         self.movement[0] = self.max_velocity
379         sleep(2)
380         self.movement[0] = 0
381         print("closed door")
382         return True
383
384     def place_object_at_destination(self, object_name: str, destination_name: str = None,
385         ↪ height_offset: int = 0.1,
386         front_offset: int = -0.05) -> None:
387         """Opens gripper and drops object on ground (optionally at destination)"""
388         if destination_name is not None:
389             dest_pos = self.resolve_object_from_name(destination_name)["pos"]
390             matrix = np.array([
391                 [0, 0, 0],
392                 [0, 1, 0],
393                 [0, 0, 0]
394             ])
395             self.move_abs(*(np.dot(self.get_eef_pos() + [0, 0, height_offset],
396                 ↪ (np.identity(3) - matrix)) + np.dot(dest_pos, matrix)))
397             self.move_abs(*(np.dot(self.get_eef_pos() + [-0.1, 0, height_offset],
398                 ↪ (np.identity(3) - matrix)) + np.dot(dest_pos, matrix)))

```

```
394         self.approach_destination_from_direction(dest_pos + [front_offset, 0,  
395             ↪ height_offset], "front")  
396     self.open_gripper()  
397     sleep(1)  
398     print(f'placed object {object_name}{f" at {destination_name}" if destination_name  
399         ↪ is not None else ""}')  
399
```

A.3. Beispiel-Log

args.json

```
1 {"batch_name": "vision_gpt-5_reasoning", "vision_enabled": true, "vision_legacy": false,
  ↪ "model": "gpt-5", "renderer": false, "log_pictures": true, "use_json": false,
  ↪ "use_reasoning": true, "send_every_tool_call": false, "use_all_functions": false,
  ↪ "use_low_level_only": false}
```

RobocasaLLM.log

```
1 Using system prompt:
2 You may only use one function call per response and have to wait for it to finish that you
  ↪ can potentially react to errors that arise during execution and are returned by the
  ↪ function. If multiple function calls are provided, only the first one will be
  ↪ executed. You will get a scene description from the user, given this description,
  ↪ think of a plan on how to achieve the task and send a message containing the plan.
  ↪ Afterwards, begin with the execution.
3
4 Using microwave prompt:
5 Here's a scene description:
6 The image shows a kitchen work area centered on a freestanding electric range with a
  ↪ smooth black glass cooktop. The cooktop has several circular heating zones and all
  ↪ are off (no indicator lights or glow). The oven door below is closed. The control
  ↪ panel behind the cooktop has four knobs; all are pointing to the off position.
7
8 Above the range is a built-in microwave. Its door is closed and has a vertical handle on
  ↪ the right and a keypad to the right of the door. A white robotic arm is positioned in
  ↪ front of the range; its gripper is near, but not holding, the microwave handle.
9
10 Left countertop (from the stove outward):
11 - A white paper towel roll on a stand, upright.
12 - A disposable coffee cup with a lid, standing closed.
13 - A wooden knife block with several knives inserted.
14 - A white plate sitting near the counter edge; it is open and holds a single whole red
  ↪ apple.
15
16 Right countertop:
17 - A two-slot toaster near the backsplash. The slots are open and empty; the lever is up
  ↪ (not toasting). The browning dial is visible.
18
19 Background/fixtures:
20 - Brick backsplash with multiple electrical outlets.
21 - Upper cabinets and lower drawers/doors are present and all closed.
22 - A dark shelf or cabinet trim runs to the right of the microwave.
23
24 Container and device states summary:
25 - Microwave: closed.
26 - Oven: closed.
27 - Cooktop: off.
28 - Toaster: empty, lever up (idle).
29 - Coffee cup: lidded (closed).
30 - Plate: open, contains one apple.
```

```

31 - Knife block: contains knives inserted.
32 - Paper towel holder: roll exposed (open).
33 - All visible cabinets/drawers: closed.
34 You are the one-armed robot with a single gripper that can only hold one thing at a time.
    ↳ Your objective is to thaw food in a microwave. The food object is called "obj" in the
    ↳ simulation, the microwave is called "container". In the end, the food should be in the
    ↳ microwave, the microwave should be turned on and you should be at least 25 cm away
    ↳ from the object. After formulating your plan, immediately begin execution.
35
36 LLM Response:
37 Plan:
38 - Verify the microwave is off and the door can be opened; keep the path clear.
39 - Open the microwave door first so I'm free to use my single gripper to fetch and place
    ↳ the food.
40 - Locate the food object named "obj": visually scan all visible surfaces; if not found,
    ↳ open nearby cabinets/drawers methodically until located.
41 - Once "obj" is found, ensure it has no obvious metal; if safe, place it on the microwave
    ↳ turntable centered.
42 - Close the microwave door and start a defrost cycle (low power) to thaw.
43 - Retract the arm to be at least 25 cm away from the object while it runs.
44
45 Execution:
46 - Move gripper to the microwave handle; align, grasp handle; pull to open the microwave
    ↳ door fully. Release handle. Verify the door stays open and the cavity is clear.
47 - Visually scan the left and right countertops for an item labeled "obj":
48   - Ignore the apple on the plate, coffee cup, paper towel roll, knife block, and toaster.
49   - If "obj" is visible on counters, note its position; if not visible, open upper
    ↳ cabinets nearest the microwave (one at a time): grasp knob/edge, pull open, look for
    ↳ "obj"; if not found, close the door and proceed to the next upper cabinet. If still
    ↳ not found, open the nearest lower drawer/door and repeat until "obj" is located.
50 - Once "obj" is located, approach it; adjust wrist for a secure grasp; close gripper to
    ↳ pick up "obj"; lift straight up to clear surrounding items.
51 - Navigate to the open microwave cavity; insert "obj" and place it centered on the
    ↳ turntable. Open gripper to release. Retract slightly to avoid contact.
52 - Reach to the microwave door edge/handle; grasp and push/close the door firmly until
    ↳ latched.
53 - On the keypad, select a thaw setting:
54   - If there is a dedicated "Defrost" button: press Defrost, then enter 3 0 0 for 3:00
    ↳ minutes, then press Start.
55   - If no Defrost: press Power Level, enter 3 (30%), enter 3 0 0 for 3:00 minutes, then
    ↳ press Start.
56 - Confirm the microwave is running (indicator on, timer counting, internal light on).
57 - Retract the arm straight back at least 30 cm from the microwave door so I am >25 cm away
    ↳ from the object inside. Hold position while the microwave runs.
58 LLM is reasoning
59
60 LLM Reasoning:
61 Plan:
62 - Verify the microwave is off and the door can be opened; keep the path clear.
63 - Open the microwave door first so I'm free to use my single gripper to fetch and place
    ↳ the food.
64 - Locate the food object named "obj": visually scan all visible surfaces; if not found,
    ↳ open nearby cabinets/drawers methodically until located.

```

```

65 - Once "obj" is found, ensure it has no obvious metal; if safe, place it on the microwave
    ↪ turntable centered.
66 - Close the microwave door and start a defrost cycle (low power) to thaw.
67 - Retract the arm to be at least 25 cm away from the object while it runs.
68
69 Execution:
70 - Move gripper to the microwave handle; align, grasp handle; pull to open the microwave
    ↪ door fully. Release handle. Verify the door stays open and the cavity is clear.
71 - Visually scan the left and right countertops for an item labeled "obj":
72   - Ignore the apple on the plate, coffee cup, paper towel roll, knife block, and toaster.
73   - If "obj" is visible on counters, note its position; if not visible, open upper
    ↪ cabinets nearest the microwave (one at a time): grasp knob/edge, pull open, look for
    ↪ "obj"; if not found, close the door and proceed to the next upper cabinet. If still
    ↪ not found, open the nearest lower drawer/door and repeat until "obj" is located.
74 - Once "obj" is located, approach it; adjust wrist for a secure grasp; close gripper to
    ↪ pick up "obj"; lift straight up to clear surrounding items.
75 - Navigate to the open microwave cavity; insert "obj" and place it centered on the
    ↪ turntable. Open gripper to release. Retract slightly to avoid contact.
76 - Reach to the microwave door edge/handle; grasp and push/close the door firmly until
    ↪ latched.
77 - On the keypad, select a thaw setting:
78   - If there is a dedicated "Defrost" button: press Defrost, then enter 3 0 0 for 3:00
    ↪ minutes, then press Start.
79   - If no Defrost: press Power Level, enter 3 (30%), enter 3 0 0 for 3:00 minutes, then
    ↪ press Start.
80 - Confirm the microwave is running (indicator on, timer counting, internal light on).
81 - Retract the arm straight back at least 30 cm from the microwave door so I am >25 cm away
    ↪ from the object inside. Hold position while the microwave runs.
82
83 LLM Response:
84 Plan:
85 - Open the microwave door so I can use my single gripper to fetch and place the food.
86 - Pick up the food object "obj" from its location.
87 - Place "obj" inside the microwave "container," centered on the turntable.
88 - Close the microwave door securely.
89 - Press the microwave button(s) to start thawing.
90 - Move the arm back to be at least 25 cm away from the object while it runs.
91
92 Beginning execution: opening the microwave door.
93 LLM wants to execute tool calls
94
95 Tool calls:
96 open_door()
97
98 LLM Response:
99 Picking up the food item "obj" to place it into the open microwave.
100 LLM wants to execute tool calls
101
102 Tool calls:
103 grip_object_from_above(object_name=obj)
104
105 LLM Response:
106 Placing the food "obj" into the microwave "container," centered on the turntable.

```

```
107 | LLM wants to execute tool calls
108 |
109 | Tool calls:
110 | place_object_at_destination(object_name=obj, destination_name=container)
111 |
112 | LLM Response:
113 | Closing the microwave door to prepare for thawing.
114 | LLM wants to execute tool calls
115 |
116 | Tool calls:
117 | close_door()
118 |
119 | LLM Response:
120 | Starting the microwave to thaw the food.
121 | LLM wants to execute tool calls
122 |
123 | Tool calls:
124 | press_button()
125 | Task accomplished successfully!
126 | SUCCESS
```

A.4. Repository

Der vollständige Quellcode sowie die Logs stehen Ihnen im folgenden Git-Repository zur Verfügung:
<https://github.com/ks3533/foundation-models-for-robotics>

B. Quellen

Literatur

- [1] brian ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as i can, not as i say: Grounding language in robotic affordances. In Karen Liu, Dana Kulic, and Jeff Ichnowski, editors, Proceedings of The 6th Conference on Robot Learning, volume 205 of Proceedings of Machine Learning Research, pages 287–318. PMLR, 14–18 Dec 2023.
- [2] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In 2023 IEEE International Conference on Robotics and Automation (ICRA), pages 9493–9500, 2023.
- [3] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In 2012 IEEE/RSJ international conference on intelligent robots and systems, pages 5026–5033. IEEE, 2012.
- [4] Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. In Robotics: Science and Systems, 2024.
- [5] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, Yifeng Zhu, and Kevin Lin. robosuite: A modular simulation framework and benchmark for robot learning. In arXiv preprint arXiv:2009.12293, 2020.
- [6] BerriAI. LiteLLM: Open-source LLM gateway and Python SDK. <https://github.com/BerriAI/litellm>, 2025. Version 1.69.2.

Selbstständigkeitserklärung

Hiermit versichern wir, dass wir diese Arbeit unter der Beratung durch Sebastian Rietsch und Timo Birr selbstständig verfasst haben und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden, sowie Zitate kenntlich gemacht wurden.

Karlsruhe, der 14. Oktober 2025

Autorin

Marxzell, der 14. Oktober 2025

Autor