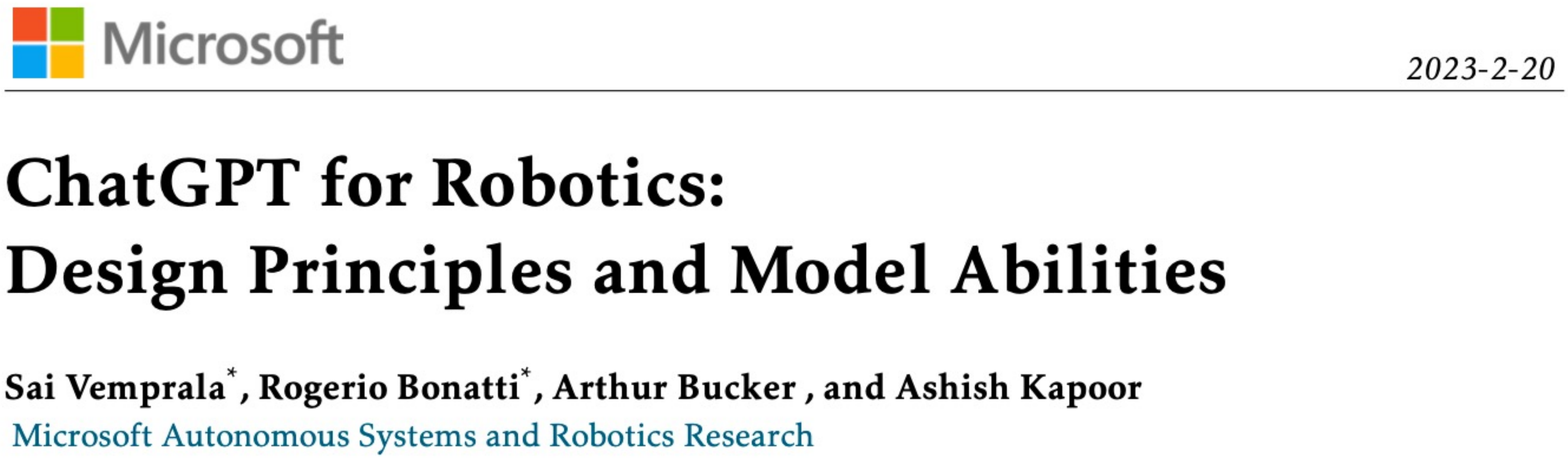
Lecture 13
ChatGPT for Robotics
(Paper Review)

Southern University of Science and Technology

Overview



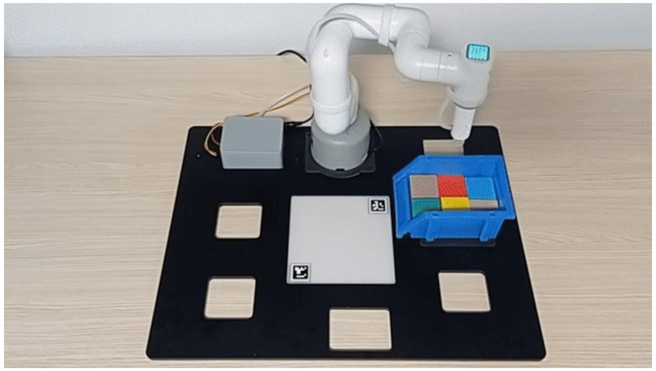
Microsoft

2023-2-20

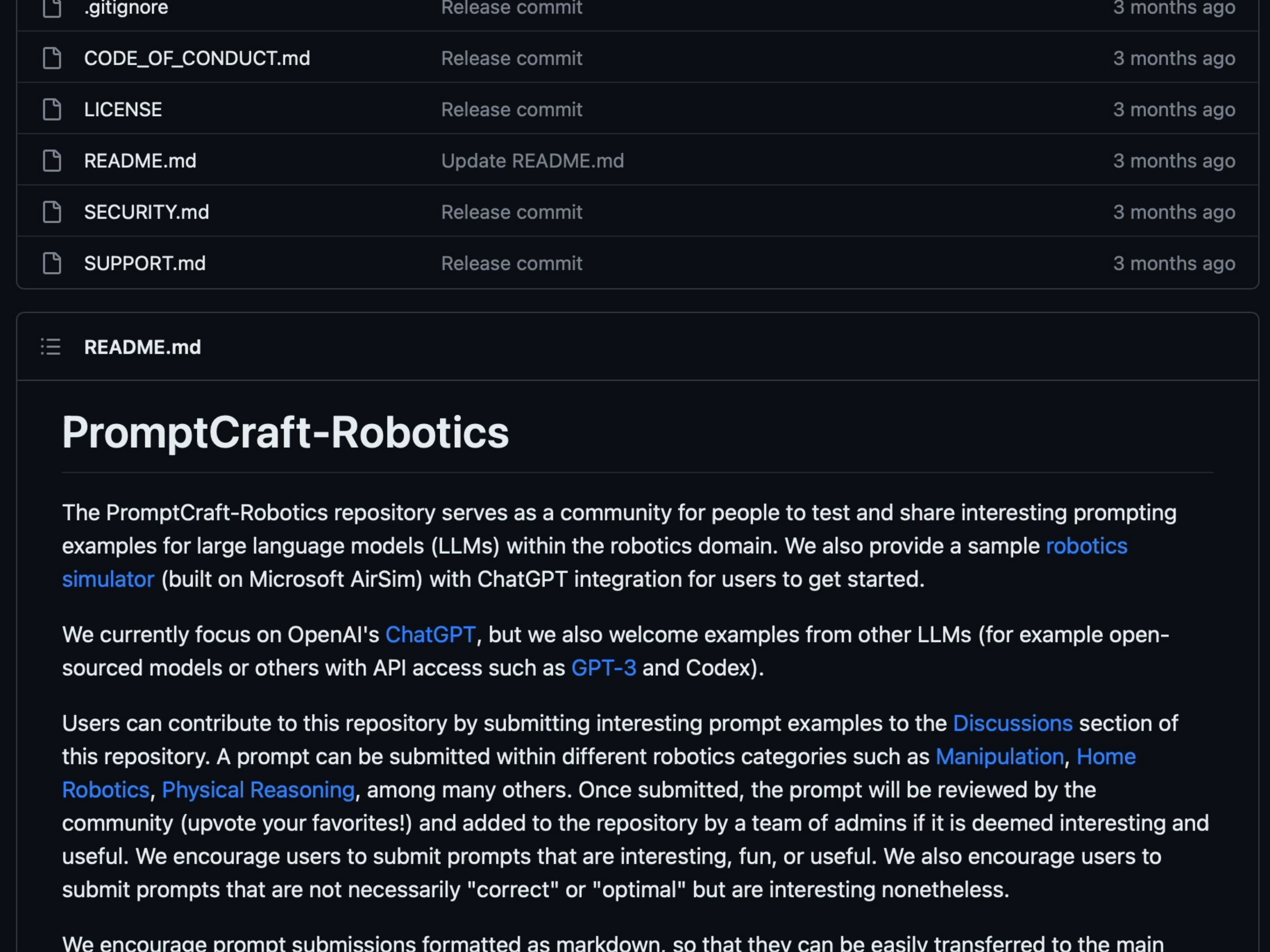
ChatGPT for Robotics: Design Principles and Model Abilities

Sai Vemprala*, Rogerio Bonatti*, Arthur Bucker, and Ashish Kapoor
Microsoft Autonomous Systems and Robotics Research

- A Technical Paper published by Microsft on 230220
  - [Blog] | [Paper] | [Code] | [Video]



- Core idea in one sentence:
  - *We extended the capabilities of ChatGPT to robotics, and controlled multiple platforms such as robot arms, drones, and home assistant robots intuitively with language.*



.gitignore	Release commit	3 months ago
CODE_OF_CONDUCT.md	Release commit	3 months ago
LICENSE	Release commit	3 months ago
README.md	Update README.md	3 months ago
SECURITY.md	Release commit	3 months ago
SUPPORT.md	Release commit	3 months ago

README.md

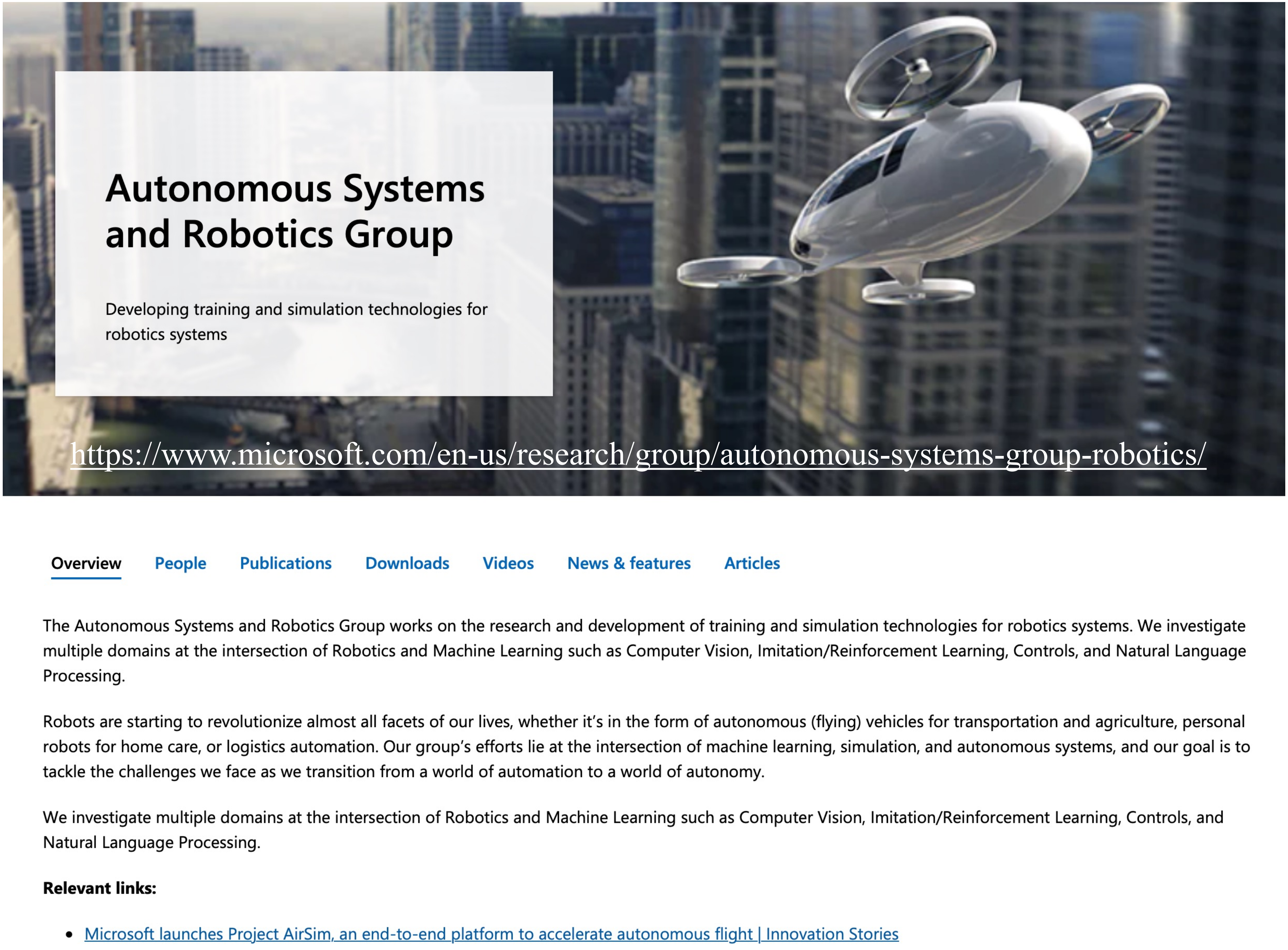
PromptCraft-Robotics

The PromptCraft-Robotics repository serves as a community for people to test and share interesting prompting examples for large language models (LLMs) within the robotics domain. We also provide a sample robotics simulator (built on Microsoft AirSim) with ChatGPT integration for users to get started.

We currently focus on OpenAI's ChatGPT, but we also welcome examples from other LLMs (for example open-sourced models or others with API access such as GPT-3 and Codex).

Users can contribute to this repository by submitting interesting prompt examples to the Discussions section of this repository. A prompt can be submitted within different robotics categories such as Manipulation, Home Robotics, Physical Reasoning, among many others. Once submitted, the prompt will be reviewed by the community (upvote your favorites!) and added to the repository by a team of admins if it is deemed interesting and useful. We encourage users to submit prompts that are interesting, fun, or useful. We also encourage users to submit prompts that are not necessarily "correct" or "optimal" but are interesting nonetheless.

We encourage prompt submissions formatted as markdown, so that they can be easily transferred to the main



Autonomous Systems and Robotics Group

Developing training and simulation technologies for robotics systems

https://www.microsoft.com/en-us/research/group/autonomous-systems-group-robotics/

The Autonomous Systems and Robotics Group works on the research and development of training and simulation technologies for robotics systems. We investigate multiple domains at the intersection of Robotics and Machine Learning such as Computer Vision, Imitation/Reinforcement Learning, Controls, and Natural Language Processing.

Robots are starting to revolutionize almost all facets of our lives, whether it's in the form of autonomous (flying) vehicles for transportation and agriculture, personal robots for home care, or logistics automation. Our group's efforts lie at the intersection of machine learning, simulation, and autonomous systems, and our goal is to tackle the challenges we face as we transition from a world of automation to a world of autonomy.

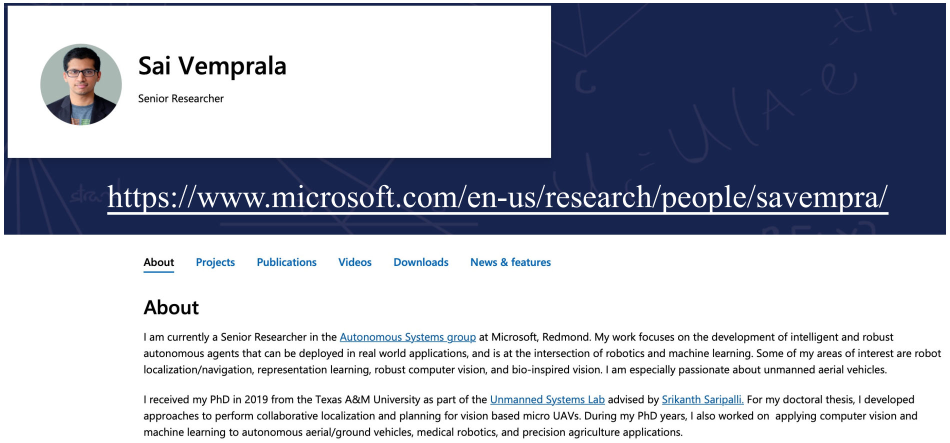
We investigate multiple domains at the intersection of Robotics and Machine Learning such as Computer Vision, Imitation/Reinforcement Learning, Controls, and Natural Language Processing.

Relevant links:

- Microsoft launches Project AirSim, an end-to-end platform to accelerate autonomous flight | Innovation Stories

About This Paper

Authors’ Info



Sai Vemprala

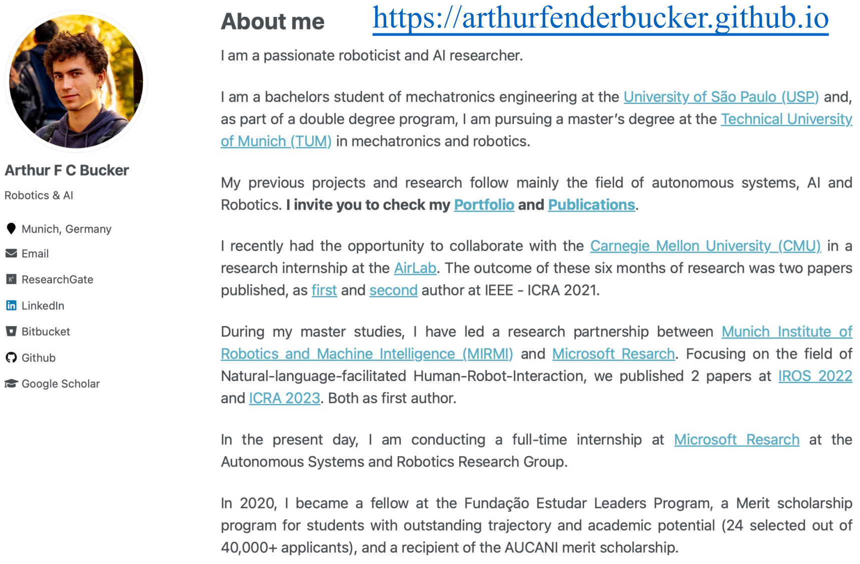
Senior Researcher

https://www.microsoft.com/en-us/research/people/savempra/

About

I am currently a Senior Researcher in the Autonomous Systems group at Microsoft, Redmond. My work focuses on the development of intelligent and robust autonomous agents that can be deployed in real world applications, and is at the intersection of robotics and machine learning. Some of my areas of interest are robot localization/navigation, representation learning, robust computer vision, and bio-inspired vision. I am especially passionate about unmanned aerial vehicles.

I received my PhD in 2019 from the Texas A&M University as part of the Unmanned Systems Lab advised by Srikanth Saripalli. For my doctoral thesis, I developed approaches to perform collaborative localization and planning for vision based micro UAVs. During my PhD years, I also worked on applying computer vision and machine learning to autonomous aerial/ground vehicles, medical robotics, and precision agriculture applications.



About me

https://arthurfenderbucker.github.io

Arthur F C Bucker

Robotics & AI

I am a passionate roboticist and AI researcher.

I am a bachelors student of mechatronics engineering at the University of São Paulo (USP) and, as part of a double degree program, I am pursuing a master’s degree at the Technical University of Munich (TUM) in mechatronics and robotics.

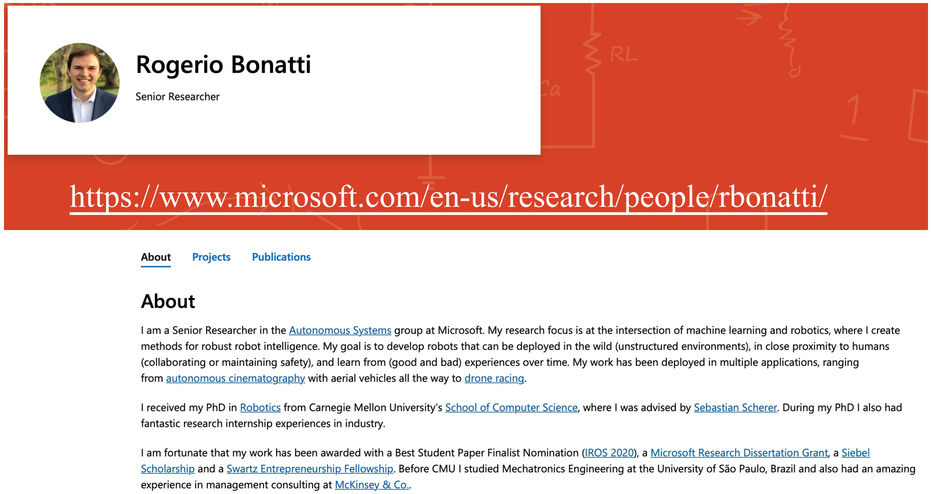
My previous projects and research follow mainly the field of autonomous systems, AI and Robotics. **I invite you to check my Portfolio and Publications.**

I recently had the opportunity to collaborate with the Carnegie Mellon University (CMU) in a research internship at the AirLab. The outcome of these six months of research was two papers published, as first and second author at IEEE - ICRA 2021.

During my master studies, I have led a research partnership between Munich Institute of Robotics and Machine Intelligence (MIRMI) and Microsoft Resarch. Focusing on the field of Natural-language-facilitated Human-Robot-Interaction, we published 2 papers at IROS 2022 and ICRA 2023. Both as first author.

In the present day, I am conducting a full-time internship at Microsoft Resarch at the Autonomous Systems and Robotics Research Group.

In 2020, I became a fellow at the Fundação Estudar Leaders Program, a Merit scholarship program for students with outstanding trajectory and academic potential (24 selected out of 40,000+ applicants), and a recipient of the AUCANI merit scholarship.



Rogerio Bonatti

Senior Researcher

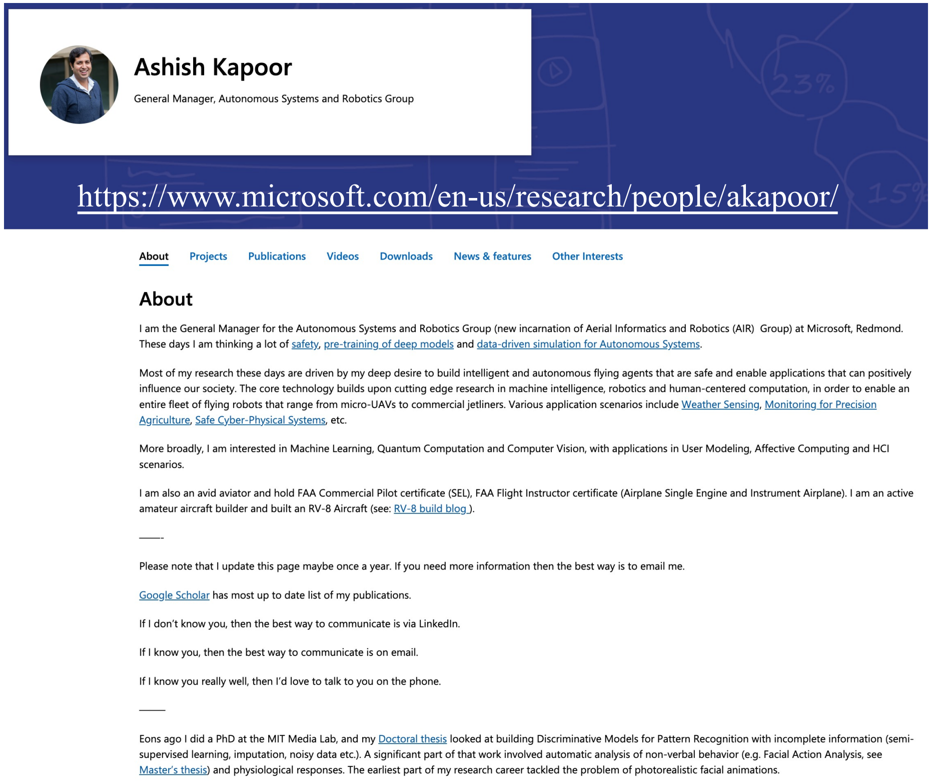
https://www.microsoft.com/en-us/research/people/rbonatti/

About

I am a Senior Researcher in the Autonomous Systems group at Microsoft. My research focus is at the intersection of machine learning and robotics, where I create methods for robust robot intelligence. My goal is to develop robots that can be deployed in the wild (unstructured environments), in close proximity to humans (collaborating or maintaining safety), and learn from (good and bad) experiences over time. My work has been deployed in multiple applications, ranging from autonomous cinematography with aerial vehicles all the way to drone racing.

I received my PhD in Robotics from Carnegie Mellon University’s School of Computer Science, where I was advised by Sebastian Scherer. During my PhD I also had fantastic research internship experiences in industry.

I am fortunate that my work has been awarded with a Best Student Paper Finalist Nomination (IROS 2020), a Microsoft Research Dissertation Grant, a Siebel Scholarship and a Swartz Entrepreneurship Fellowship. Before CMU I studied Mechatronics Engineering at the University of São Paulo, Brazil and also had an amazing experience in management consulting at McKinsey & Co..



Ashish Kapoor

General Manager, Autonomous Systems and Robotics Group

https://www.microsoft.com/en-us/research/people/akapoor/

About

I am the General Manager for the Autonomous Systems and Robotics Group (new incarnation of Aerial Informatics and Robotics (AIR) Group) at Microsoft, Redmond. These days I am thinking a lot of safety, pre-training of deep models and data-driven simulation for Autonomous Systems.

Most of my research these days are driven by my deep desire to build intelligent and autonomous flying agents that are safe and enable applications that can positively influence our society. The core technology builds upon cutting edge research in machine intelligence, robotics and human-centered computation, in order to enable an entire fleet of flying robots that range from micro-UAVs to commercial jetliners. Various application scenarios include Weather Sensing, Monitoring for Precision Agriculture, Safe Cyber-Physical Systems, etc.

More broadly, I am interested in Machine Learning, Quantum Computation and Computer Vision, with applications in User Modeling, Affective Computing and HCI scenarios.

I am also an avid aviator and hold FAA Commercial Pilot certificate (SEL), FAA Flight Instructor certificate (Airplane Single Engine and Instrument Airplane). I am an active amateur aircraft builder and built an RV-8 Aircraft (see: RV-8 build blog).

——-

Please note that I update this page maybe once a year. If you need more information then the best way is to email me.

Google Scholar has most up to date list of my publications.

If I don’t know you, then the best way to communicate is via LinkedIn.

If I know you, then the best way to communicate is on email.

If I know you really well, then I’d love to talk to you on the phone.

———

Eons ago I did a PhD at the MIT Media Lab, and my Doctoral thesis looked at building Discriminative Models for Pattern Recognition with incomplete information (semi-supervised learning, imputation, noisy data etc.). A significant part of that work involved automatic analysis of non-verbal behavior (e.g. Facial Action Analysis, see Master’s thesis) and physiological responses. The earliest part of my research career tackled the problem of photorealistic facial animations.

Abstract (TL;DR)

- This paper presents an experimental study regarding the use of OpenAI’s ChatGPT [1] for robotics applications. We outline a strategy that combines design principles for prompt engineering and the creation of a high-level function library which allows ChatGPT to adapt to different robotics tasks, simulators, and form factors. We focus our evaluations on the effectiveness of different prompt engineering techniques and dialog strategies towards the execution of various types of robotics tasks. We explore ChatGPT’s ability to use free-form dialog, parse XML tags, and to synthesize code, in addition to the use of task-specific prompting functions and closed-loop reasoning through dialogues. Our study encompasses a range of tasks within the robotics domain, from basic logical, geometrical, and mathematical reasoning all the way to complex domains such as aerial navigation, manipulation, and embodied agents. We show that ChatGPT can be effective at solving several of such tasks, while allowing users to interact with it primarily via natural language instructions. In addition to these studies, we introduce an open-sourced research tool called PromptCraft, which contains a platform where researchers can collaboratively upload and vote on examples of good prompting schemes for robotics applications, as well as a sample robotics simulator with ChatGPT integration, making it easier for users to get started with using ChatGPT for robotics.

Abstract (The Decomposed Version)

A short and brief summary	This paper presents an experimental study regarding the use of OpenAI’s ChatGPT [1] for robotics applications.
Elaboration on the proposed method (challenge)	We outline a strategy that combines design principles for prompt engineering and the creation of a high-level function library which allows ChatGPT to adapt to different robotics tasks, simulators, and form factors.
Aim of this paper, how it’s different?	We focus our evaluations on the effectiveness of different prompt engineering techniques and dialog strategies towards the execution of various types of robotics tasks.
What has been done?	We explore ChatGPT’s ability to use free-form dialog, parse XML tags, and to synthesize code, in addition to the use of task-specific prompting functions and closed-loop reasoning through dialogues.
Elaborate on why what’s been done	Our study encompasses a range of tasks within the robotics domain, from basic logical, geometrical, and mathematical reasoning all the way to complex domains such as aerial navigation, manipulation, and embodied agents.
What are the outcomes of this work (qualitative description here)	We show that ChatGPT can be effective at solving several of such tasks, while allowing users to interact with it primarily via natural language instructions.
What else that is important in this work Why they are also important?	In addition to these studies, we introduce an open-sourced research tool called PromptCraft, which contains a platform where researchers can collaboratively upload and vote on examples of good prompting schemes for robotics applications, as well as a sample robotics simulator with ChatGPT integration, making it easier for users to get started with using ChatGPT for robotics.

About This Paper

Overview

- 25 pages in total => long

- 16 pages for main content
- 34 cited references
- 6 pages for appendix => long

- Unusual paper structure with a short **Related Work** before **Conclusion**

- Many video demos

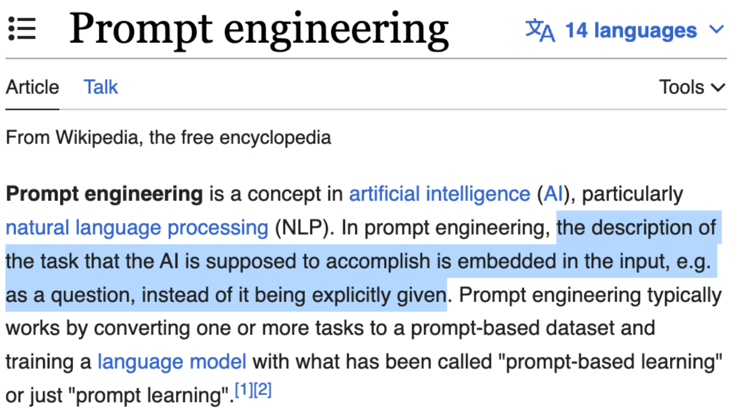
- Sec. 2 on Core Method
  - A design guideline
- Sec. 3 on Results & Demos
  - With a brief discussion
- Sec. 4 on a New Tool
  - Short and brief

- 1. Introduction
- 2. Robotics with ChatGPT
  - 2.1. Construction and description of the robotics API library
  - 2.2. Clear description of the task details in the prompt
  - 2.3. Special arguments to bias the answer's structure
- 3. ChatGPT abilities for solving robotics problems
  - 3.1. Zero-shot task planning
    - 3.1.1. Spatio-temporal reasoning: catching a basketball with visual servoing
    - 3.1.2. Aerial robotics: real-world drone flight with intuitive human-robot interface (Click for video)
    - 3.1.3. Aerial robotics: AirSim industrial inspection (Click for video)
  - 3.2. User on the loop: interactive conversations for complex tasks
    - 3.2.1. Manipulation with curriculum learning (Click for video)
    - 3.2.2. Aerial robotics: AirSim obstacle avoidance (Click for video)
  - 3.3. Perception-action loops
    - 3.3.1. Embodied agent: closed loop object navigation with API library (Click for video)
    - 3.3.2. Embodied agent: closed loop visual-language navigation using ChatGPT's dialogue (Click for video)
  - 3.4. Reasoning and common-sense robotics tasks
- 4. PromptCraft, a collaborative tool for LLM + Robotics research
- 5. Related Work
- 6. Conclusions and Future Work
  - 6.1. ChatGPT for paper writing
- References
- A. Additional examples on design principles on prompt engineering
- B. Original initial prompts for robotics tasks examples
- C. Additional experiments
- D. Other commonsense/reasoning tasks

Motivation and Main Problem

To investigate if and how the abilities of ChatGPT can generalize to the domain of robotics

- Natural Language Processing (NLP) => Large Language Models (LLMs)
  - **Examples**: BERT, GPT-3, and Codex
  - **Applications**: text generation, machine translation, and code synthesis
  - **Limitation**: operate mostly upon a single ***prompt***



Prompt engineering

From Wikipedia, the free encyclopedia

Prompt engineering is a concept in artificial intelligence (AI), particularly natural language processing (NLP). In prompt engineering, the description of the task that the AI is supposed to accomplish is embedded in the input, e.g. as a question, instead of it being explicitly given. Prompt engineering typically works by converting one or more tasks to a prompt-based dataset and training a language model with what has been called "prompt-based learning" or just "prompt learning".[1][2]

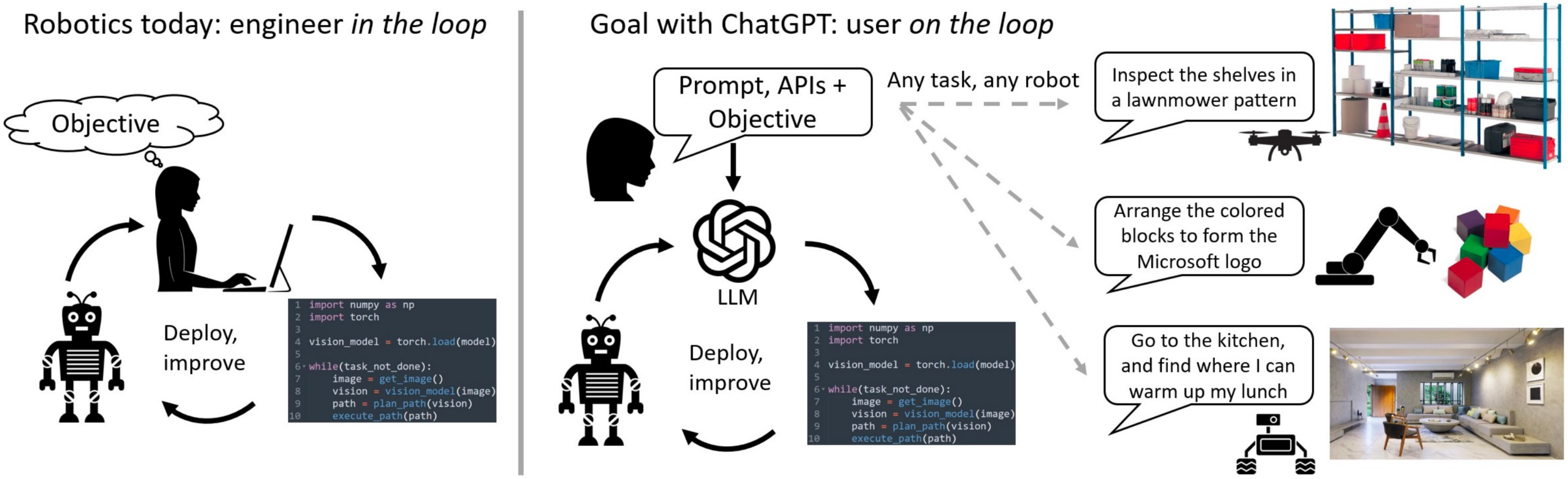
- **A New Model**: OpenAI ChatGPT
  - **All-in-one benfit**: provides particularly impressive interaction skills through **dialog**, combining text generation with code synthesis

- **Robotic** systems
  - Require a deep understanding of real-world physics, envi- ronmental context, and the ability to perform physical actions.

- A **generative** robotics model
  - a robust **commonsense knowledge**
  - a sophisticated **world model**
  - **the ability to interact with users** to interpret and execute commands in ways that are physically possible and that makes sense in the real world

- **The Challenge**
  - These challenges fall **beyond** the original scope of language models,
  - as they must not only **understand the meaning of a given text**,
  - but also **translate the intent into a logical sequence of physical actions**.

Prompting LLMs for robotics control



Robotics today: engineer *in the loop*
Objective
Deploy, improve
Goal with ChatGPT: user *on the loop*
Prompt, APIs + Objective
LLM
Deploy, improve
Any task, any robot
Inspect the shelves in a lawnmower pattern
Arrange the colored blocks to form the Microsoft logo
Go to the kitchen, and find where I can warm up my lunch

```
import numpy as np
import torch

vision_model = torch.load(model)

while(task_not_done):
    image = get_image()
    vision = vision_model(image)
    path = plan_path(vision)
    execute_path(path)
```

```
import numpy as np
import torch

vision_model = torch.load(model)

while(task_not_done):
    image = get_image()
    vision = vision_model(image)
    path = plan_path(vision)
    execute_path(path)
```

Figure 1: Current robotics pipelines require a specialized engineer *in the loop* to write code to improve the process. Our goal with ChatGPT is to have a (potentially non-technical) user *on the loop*, interacting with the language model through high-level language commands, and able to seamlessly deploy various platforms and tasks.

- With **a specialized engineer** *in the loop*, or
- With **ChatPGT** to enable a (potentially non-technical) user *on the loop*
- To seamlessly deploy various robotic platforms and tasks?

Lower the entry barrier of interaction in robotics for mass users (or workers)

Natural language and robotics

- **A crucial component for human-robot interaction.**
    - Many applications where robots can benefit from NLP
    - i.e., task instruction, navigation, and information retrieval.

- **Classically**, modeling human-robot interactions using language is challenging
    - Forces the user to operate within a rigid set of instructions
    - Requires mathematically complex algorithms to keep track of multiple probability distributions over actions and target objects

- **More recent works** explore neural networks to implicitly keep track of the complex mapping between language and actions
    - Often require vast amounts of labeled data for training

Large (vision and) language models for robotics

- **The Transformer architecture**
  - Robot control and planning, object recognition, and robot navigation
  - As feature extraction modules for one or more modalities simultaneously.
  - These systems are often coupled with additional features from pretrained large-scale vision and language models models

- **Make use of prompting structures** with pre-defined functions, behaviors, and examples to guide the generation of the model's answers
  - SayCan: focus on grounding LLMs so that free-form text commands are used to compute a value function to rank the best action types within a robot-specific library.
  - RT-1: takes an end-to-end approach to learn the mapping between language commands low level actions, without the use of intermediate high-level functions.

- Conceptually, **the main difference is the conversational ability of LLM**
  - This work allows the user to interactively improve and correct the robot's behavior (as opposed to re-engineering the prompt from scratch and generating another zero-shot answer).
  - This work aims to provide a generalizable pipeline and set of principles to be used by researchers in different fields of robotics, as opposed to focusing on a single domain such as table-top manipulation or task planning.

Prompting LLMs with APIs, and its connections to symbolic AI

- When designing LLM prompts for robotics applications, users often make use of **high-level library of APIs to represent specific behaviors to be used.**
    - We can draw a connection between this approach with classical symbolic AI, which **uses logic and rules to represent and reason about knowledge**.
    - While the traditional symbolic AI approach presented difficulties in new knowledge acquisition and dealing with out-of-distribution data, **we believe that LLMs can overcome these challenges.**
- Models such as ChatGPT **can compose new primitive functions based on the context and generate code for them automatically**.

Pipeline, Platform & Sim Env.

- **LLMs side**: The following problems are to be addressed
  - Providing a complete and accurate **descriptions of the problem**,
  - Identifying the right set of **allowable function calls and APIs**, and
  - **Biasing the answer structure** with special arguments

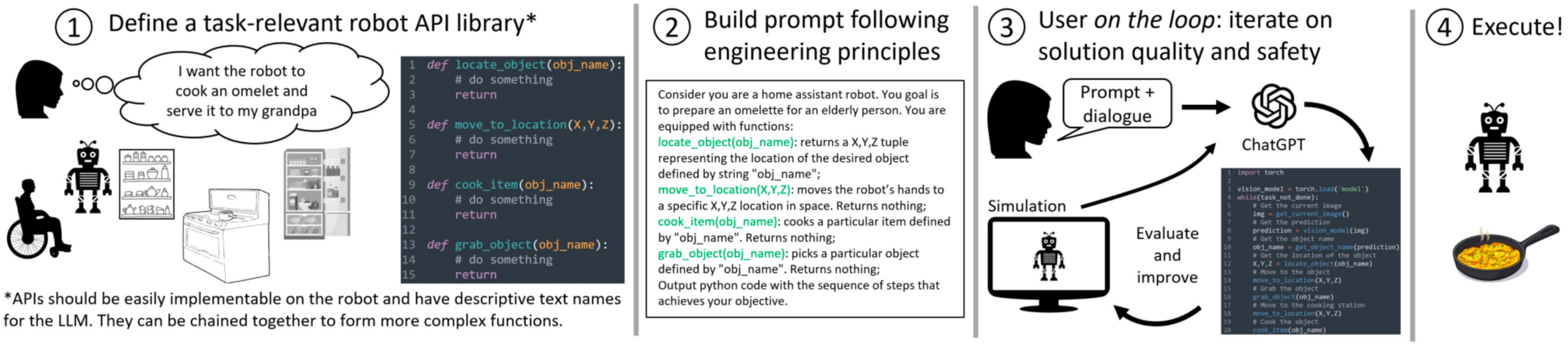
  - **Proposed Solution 1** => A *4-step* working pipeline

- **Robotics side**: Deployment issues in robotics applications
  - How to collaboratively develop prompts for robotics?
  - **Proposed Solution 2** => An opensource platform as a tool: *PromptCraft*

  - How to provide a simulation environment for LLMs with Robotics?
  - **Proposed Solution 3** => A simulation environment as a tool: *AirSim-ChatGPT*

The Proposed Pipeline / Workflow

- **1. First, we define a high-level robot function library.** This library can be specific to the form factor or scenario of interest, and should map to actual implementations on the robot platform while being named descriptively enough for ChatGPT to follow;
- **2. Next, we build a prompt for ChatGPT** which describes the objective while also identifying the set of allowed high-level functions from the library. The prompt can also contain information about constraints, or how ChatGPT should structure its responses;
- **3. The user stays on the loop to evaluate code output by ChatGPT**, either through direct analysis or through simulation, and provides feedback to ChatGPT on the quality and safety of the output code;
- **4.** After iterating on the ChatGPT-generated implementations, **the final code can be deployed onto the robot.**


① Define a task-relevant robot API library*
I want the robot to cook an omelet and serve it to my grandpa
② Build prompt following engineering principles
③ User *on the loop*: iterate on solution quality and safety
Prompt + dialogue
ChatGPT
Simulation
Evaluate and improve
④ Execute!

```
def locate_object(obj_name):
    # do something
    return

def move_to_location(X,Y,Z):
    # do something
    return

def cook_item(obj_name):
    # do something
    return

def grab_object(obj_name):
    # do something
    return
```

Consider you are a home assistant robot. You goal is to prepare an omelette for an elderly person. You are equipped with functions:
locate_object(obj_name): returns a X,Y,Z tuple representing the location of the desired object defined by string "obj_name";
move_to_location(X,Y,Z): moves the robot's hands to a specific X,Y,Z location in space. Returns nothing;
cook_item(obj_name): cooks a particular item defined by "obj_name". Returns nothing;
grab_object(obj_name): picks a particular object defined by "obj_name". Returns nothing;
Output python code with the sequence of steps that achieves your objective.

```
import torch

vision_model = torch.load('model')
while(task_not_done):
    # Get the current image
    img = get_current_image()
    # Get the prediction
    prediction = vision_model(img)
    # Get the object name
    obj_name = get_object_name(prediction)
    # Get the location of the object
    X,Y,Z = locate_object(obj_name)
    # Move to the object
    move_to_location(X,Y,Z)
    # Grab the object
    grab_object(obj_name)
    # Move to the cooking station
    move_to_location(X,Y,Z)
    # Cook the object
    cook_item(obj_name)
```

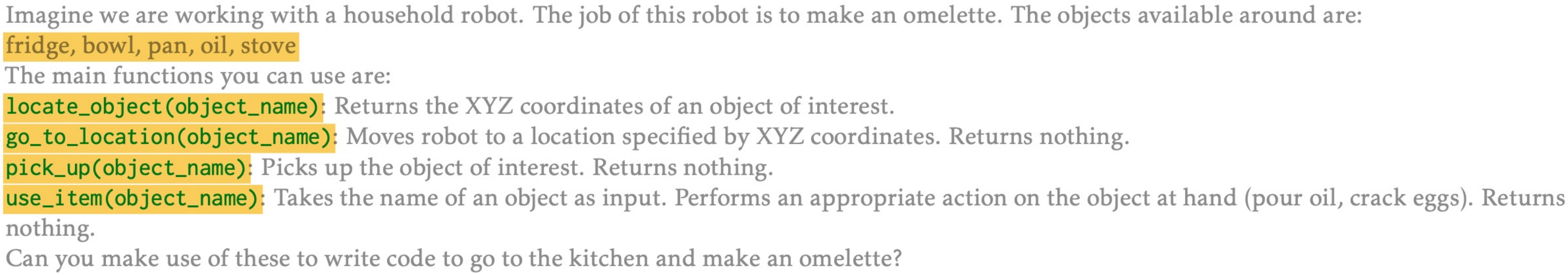
*APIs should be easily implementable on the robot and have descriptive text names for the LLM. They can be chained together to form more complex functions.

Figure 2: Robotics pipeline employing ChatGPT with the user *on the loop* to evaluate the output's quality and safety.

Proposed Approach / Algorithm / Method

Construction and description of the robotics API library

- All API names must be descriptive of the overall function behavior
    - Define high-level functions, which act as wrappers over actual implementations from the respective libraries
    - **Listing such a collection of high-level functions in the prompt is key**
        - in allowing ChatGPT to create logical sequences of behavioral primitives, and
        - in generalizing to different scenarios and platforms.

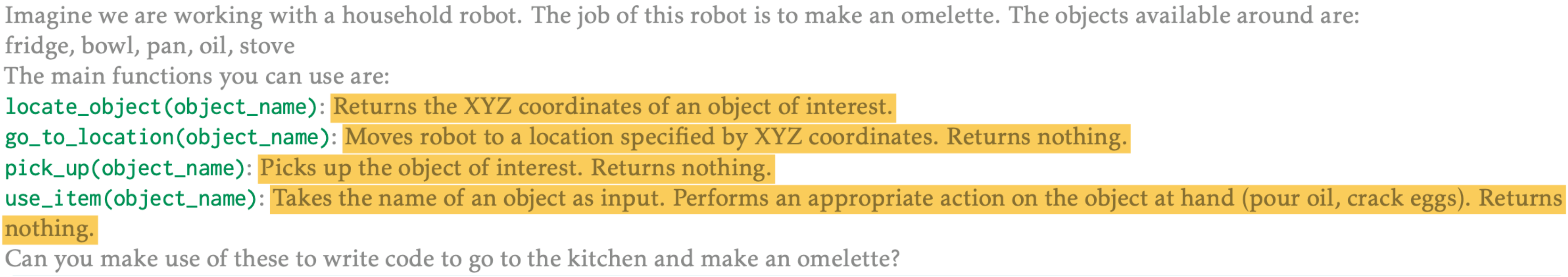


Imagine we are working with a household robot. The job of this robot is to make an omelette. The objects available around are:
fridge, bowl, pan, oil, stove
The main functions you can use are:
`locate_object(object_name)`: Returns the XYZ coordinates of an object of interest.
`go_to_location(object_name)`: Moves robot to a location specified by XYZ coordinates. Returns nothing.
`pick_up(object_name)`: Picks up the object of interest. Returns nothing.
`use_item(object_name)`: Takes the name of an object as input. Performs an appropriate action on the object at hand (pour oil, crack eggs). Returns nothing.
Can you make use of these to write code to go to the kitchen and make an omelette?

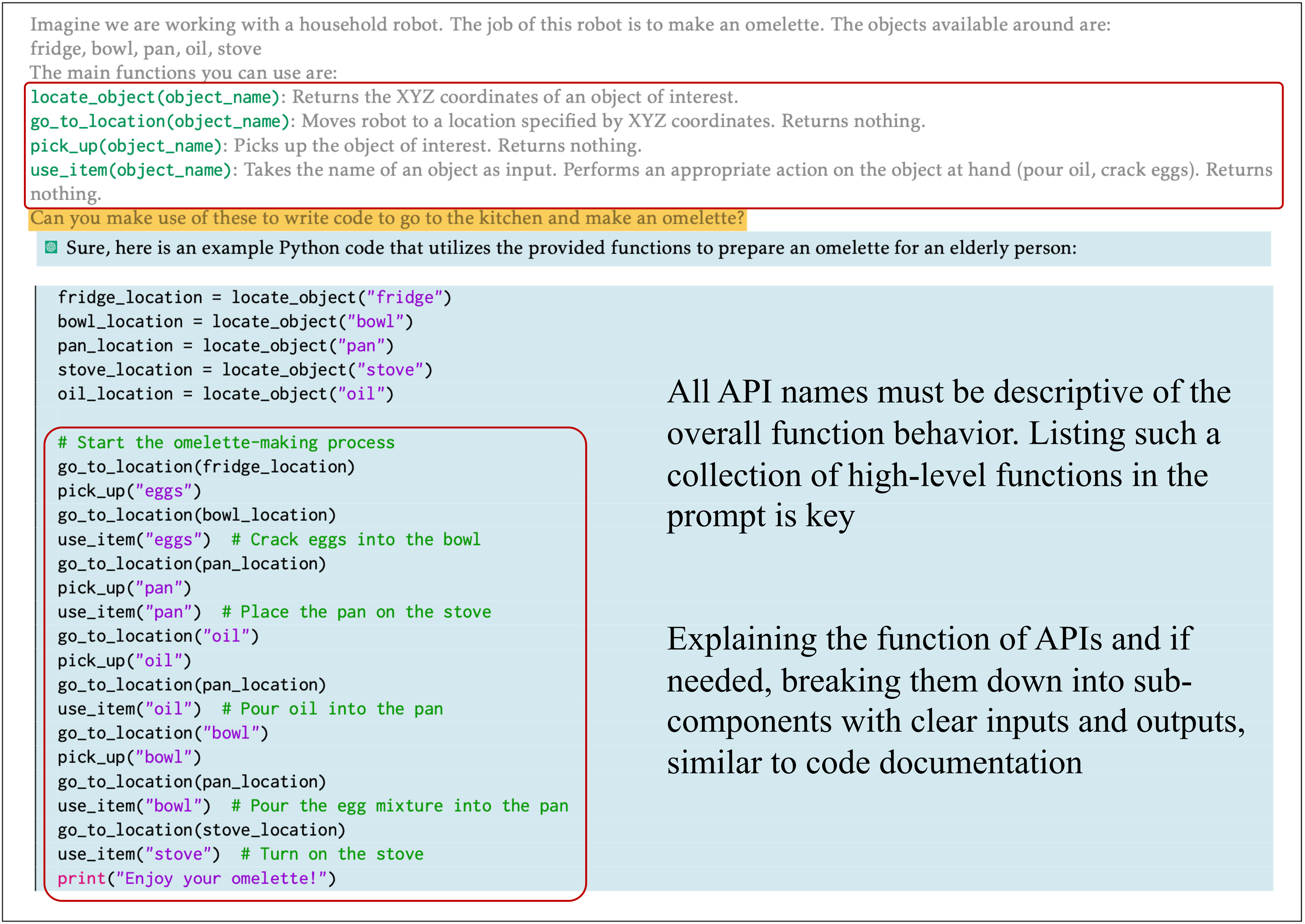
        - a function named **detect_object(object_name)** could internally link to an OpenCV function or a computer vision model,
        - something like **move_to(x, y, z)** could internally invoke a motion planning and obstacle avoidance pipeline along with the appropriate low-level motor commands for a drone.

Construction and description of the robotics API library

- Explaining the function of APIs and if needed, breaking them down into sub-components with clear inputs and outputs, similar to code documentation
    - Recommended
    - The strategy presented allows ChatGPT to reason about the order and content of tasks according to the functions the robot is actually able to execute.



Imagine we are working with a household robot. The job of this robot is to make an omelette. The objects available around are:
fridge, bowl, pan, oil, stove
The main functions you can use are:
`locate_object(object_name)`: Returns the XYZ coordinates of an object of interest.
`go_to_location(object_name)`: Moves robot to a location specified by XYZ coordinates. Returns nothing.
`pick_up(object_name)`: Picks up the object of interest. Returns nothing.
`use_item(object_name)`: Takes the name of an object as input. Performs an appropriate action on the object at hand (pour oil, crack eggs). Returns nothing.
Can you make use of these to write code to go to the kitchen and make an omelette?



Imagine we are working with a household robot. The job of this robot is to make an omelette. The objects available around are: fridge, bowl, pan, oil, stove

The main functions you can use are:

`locate_object(object_name)`: Returns the XYZ coordinates of an object of interest.

`go_to_location(object_name)`: Moves robot to a location specified by XYZ coordinates. Returns nothing.

`pick_up(object_name)`: Picks up the object of interest. Returns nothing.

`use_item(object_name)`: Takes the name of an object as input. Performs an appropriate action on the object at hand (pour oil, crack eggs). Returns nothing.

Can you make use of these to write code to go to the kitchen and make an omelette?

Sure, here is an example Python code that utilizes the provided functions to prepare an omelette for an elderly person:

```
fridge_location = locate_object("fridge")
bowl_location = locate_object("bowl")
pan_location = locate_object("pan")
stove_location = locate_object("stove")
oil_location = locate_object("oil")

# Start the omelette-making process
go_to_location(fridge_location)
pick_up("eggs")
go_to_location(bowl_location)
use_item("eggs")  # Crack eggs into the bowl
go_to_location(pan_location)
pick_up("pan")
use_item("pan")  # Place the pan on the stove
go_to_location("oil")
pick_up("oil")
go_to_location(pan_location)
use_item("oil")  # Pour oil into the pan
go_to_location("bowl")
pick_up("bowl")
go_to_location(pan_location)
use_item("bowl")  # Pour the egg mixture into the pan
go_to_location(stove_location)
use_item("stove")  # Turn on the stove
print("Enjoy your omelette!")
```

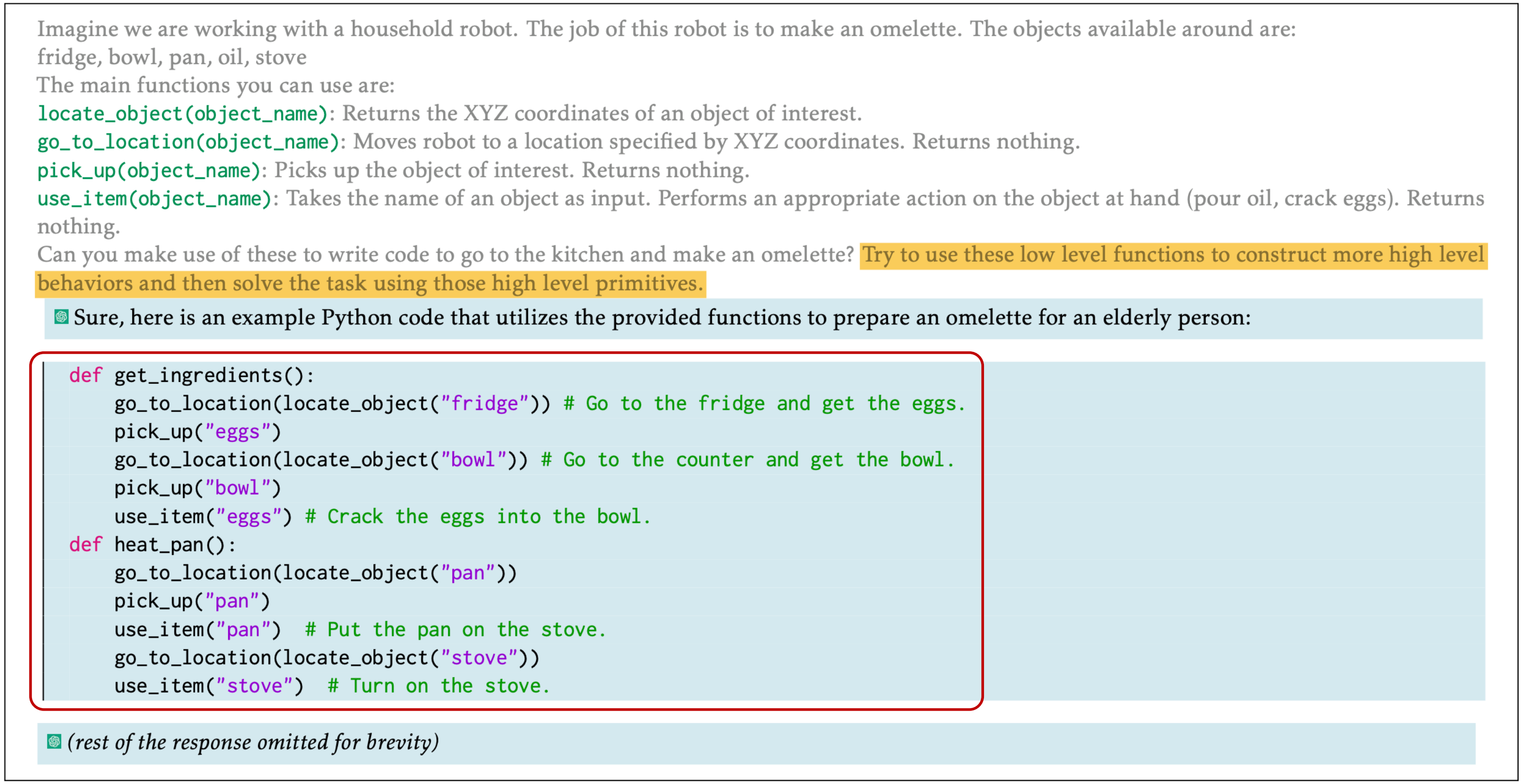
All API names must be descriptive of the overall function behavior. Listing such a collection of high-level functions in the prompt is key

Explaining the function of APIs and if needed, breaking them down into sub-components with clear inputs and outputs, similar to code documentation

Figure 3: Example of a good prompting practice when defining different APIs for a robotics problem. Clear function definitions that can be executed by the robot, and the task objectives are explicitly stated.

Construction and description of the robotics API library

- **LLMs are capable of defining new functions and concepts** altogether when relevant to a particular problem when dealing with robotics applications



Imagine we are working with a household robot. The job of this robot is to make an omelette. The objects available around are:
fridge, bowl, pan, oil, stove
The main functions you can use are:
`locate_object(object_name)`: Returns the XYZ coordinates of an object of interest.
`go_to_location(object_name)`: Moves robot to a location specified by XYZ coordinates. Returns nothing.
`pick_up(object_name)`: Picks up the object of interest. Returns nothing.
`use_item(object_name)`: Takes the name of an object as input. Performs an appropriate action on the object at hand (pour oil, crack eggs). Returns nothing.
Can you make use of these to write code to go to the kitchen and make an omelette? Try to use these low level functions to construct more high level behaviors and then solve the task using those high level primitives.

Sure, here is an example Python code that utilizes the provided functions to prepare an omelette for an elderly person:

```
def get_ingredients():
    go_to_location(locate_object("fridge")) # Go to the fridge and get the eggs.
    pick_up("eggs")
    go_to_location(locate_object("bowl")) # Go to the counter and get the bowl.
    pick_up("bowl")
    use_item("eggs") # Crack the eggs into the bowl.
def heat_pan():
    go_to_location(locate_object("pan"))
    pick_up("pan")
    use_item("pan")  # Put the pan on the stove.
    go_to_location(locate_object("stove"))
    use_item("stove")  # Turn on the stove.
```

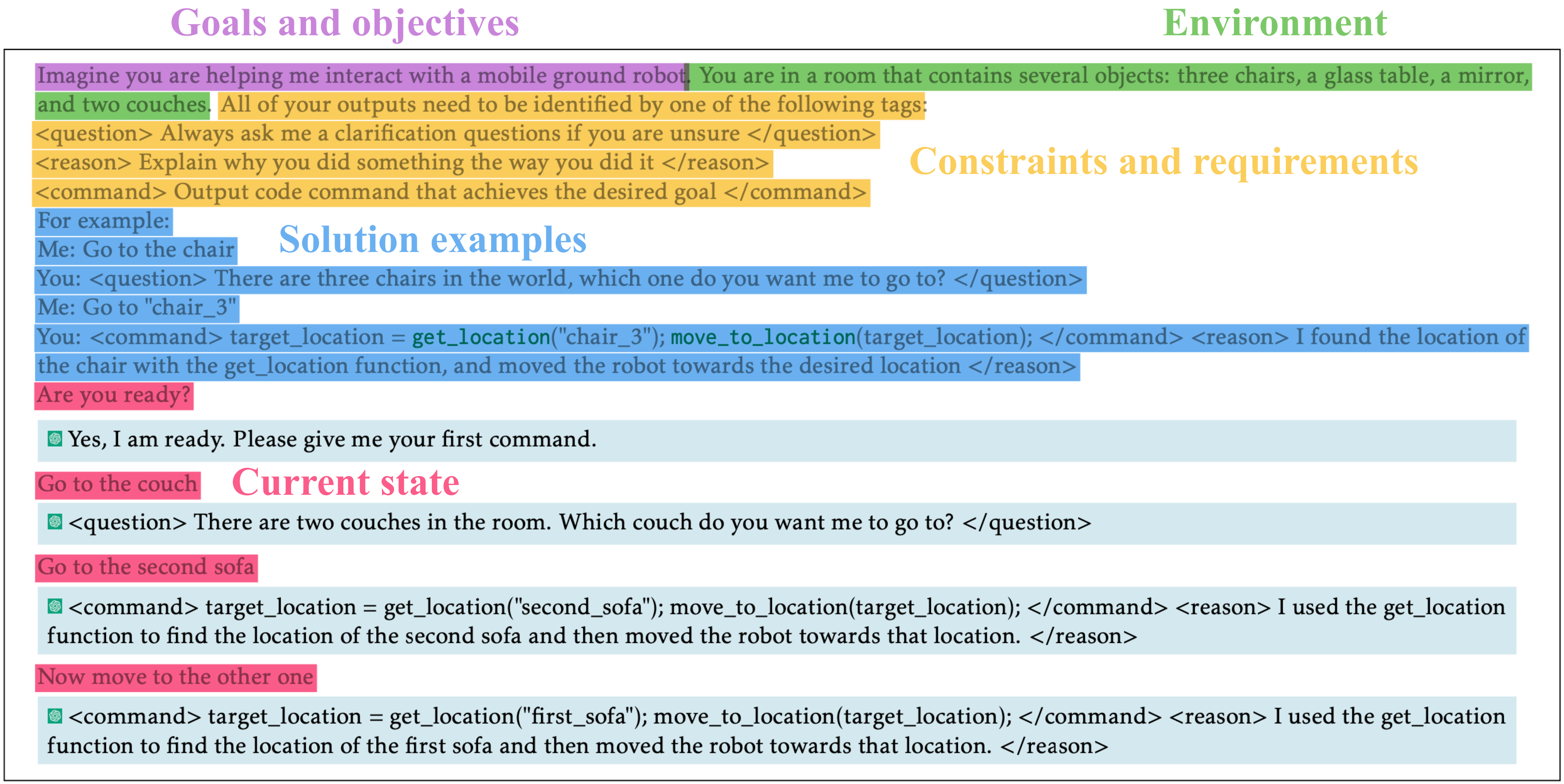
(rest of the response omitted for brevity)

Figure 4: Example of new API development using LLM intrinsic reasoning capabilities. The model improves its answer in comparison to Figure 3, which contained the same task but with a fixed set of APIs.

Clear description of the task details in the prompt

- **A good context description should contain**, besides robot APIs:
  - **Constraints and requirements**: specify constraints or requirements that are relevant to the task. If the task involves moving objects, you might specify the weight, size, and shape of the objects to be moved.
  - **Environment**: describe the environment in which the robotics task is taking place. For example, if the task is to navigate a maze, you might describe the size and shape of the maze, as well as any obstacles or hazards that need to be avoided.
  - **Current state**: describe the current state of the robotics system. For example, if the task is to pick up an object, you might describe the current position and orientation of the robot and the object.
  - **Goals and objectives**: state the goals and objectives of the task. If the task is to assemble a puzzle, you might specify the number of pieces that need to be assembled and the desired completion time.
  - **Solution examples**: demonstrate how similar tasks can be solved as a means to guide the LLM's solution strategy. For example, if a task involves interactions with the user, we can describe an example of how and when the robot should be asking for the user's input (see Fig. 5). Note that priming can also introduce biases, so we should provide a diverse range of examples and avoid overly prescriptive language.

Clear description of the task details in the prompt



Imagine you are helping me interact with a mobile ground robot. You are in a room that contains several objects: three chairs, a glass table, a mirror, and two couches. All of your outputs need to be identified by one of the following tags:
<question> Always ask me a clarification questions if you are unsure </question>
<reason> Explain why you did something the way you did it </reason>
<command> Output code command that achieves the desired goal </command>
For example:
Me: Go to the chair
You: <question> There are three chairs in the world, which one do you want me to go to? </question>
Me: Go to "chair_3"
You: <command> target_location = `get_location`("chair_3"); `move_to_location`(target_location); </command> <reason> I found the location of the chair with the get_location function, and moved the robot towards the desired location </reason>
Are you ready?

Yes, I am ready. Please give me your first command.

Go to the couch

<question> There are two couches in the room. Which couch do you want me to go to? </question>

Go to the second sofa

<command> target_location = get_location("second_sofa"); move_to_location(target_location); </command> <reason> I used the get_location function to find the location of the second sofa and then moved the robot towards that location. </reason>

Now move to the other one

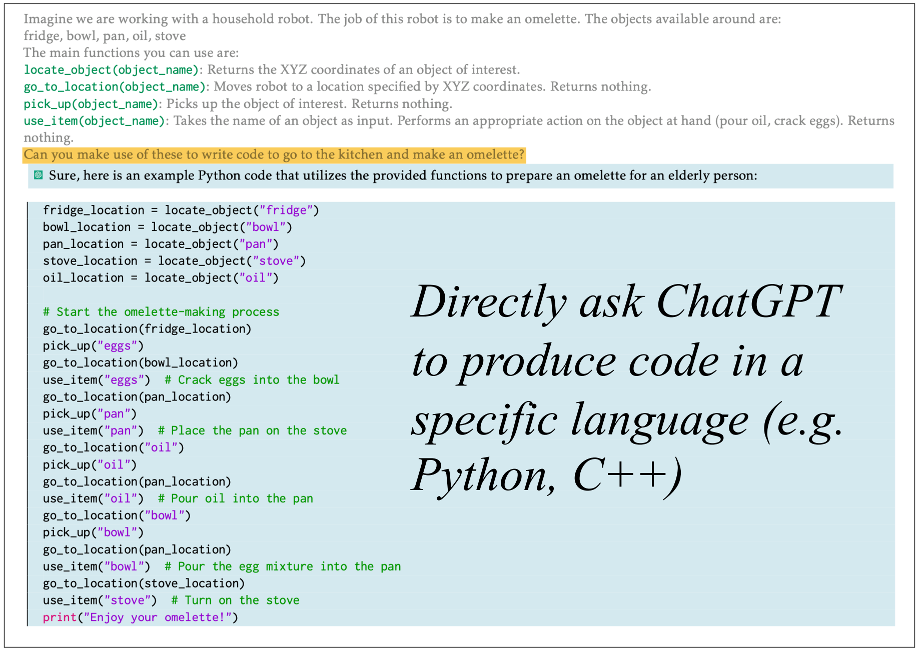
<command> target_location = get_location("first_sofa"); move_to_location(target_location); </command> <reason> I used the get_location function to find the location of the first sofa and then moved the robot towards that location. </reason>

Figure 5: Example of how user prompting can bias the model's response. We demonstrate how and when to interact with the user to ask for help. In addition, we force responses to be framed into specific XML tags which can aid code parsing in a post-processing stage.

Proposed Approach / Algorithm / Method

Special arguments to bias the answer’s structure

- Different prompting methodologies can be used to force the output of the model to obey some specific pattern.
  - A user might want to automatically parse ChatGPT’s output so that it can be used in other scripts for real-time execution



Imagine we are working with a household robot. The job of this robot is to make an omelette. The objects available around are:
fridge, bowl, pan, oil, stove
The main functions you can use are:
`locate_object(object_name)`: Returns the XYZ coordinates of an object of interest.
`go_to_location(object_name)`: Moves robot to a location specified by XYZ coordinates. Returns nothing.
`pick_up(object_name)`: Picks up the object of interest. Returns nothing.
`use_item(object_name)`: Takes the name of an object as input. Performs an appropriate action on the object at hand (pour oil, crack eggs). Returns nothing.
Can you make use of these to write code to go to the kitchen and make an omelette?

Sure, here is an example Python code that utilizes the provided functions to prepare an omelette for an elderly person:

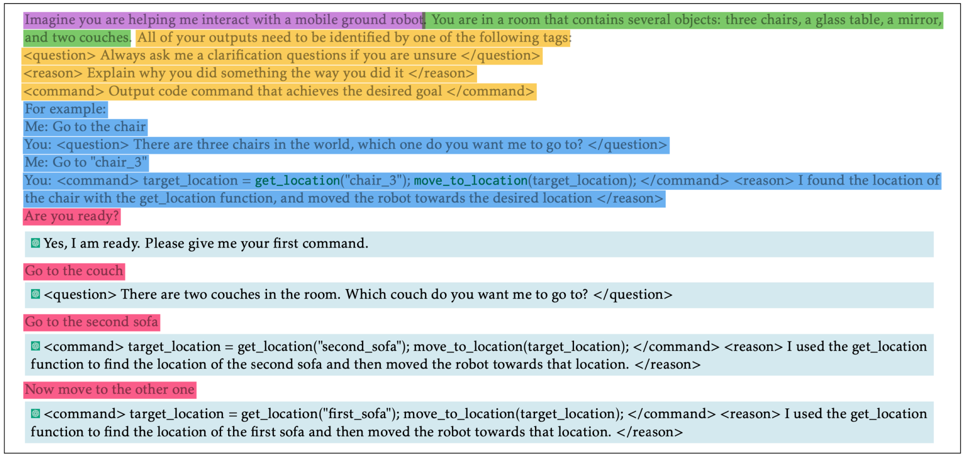
```
fridge_location = locate_object("fridge")
bowl_location = locate_object("bowl")
pan_location = locate_object("pan")
stove_location = locate_object("stove")
oil_location = locate_object("oil")

# Start the omelette-making process
go_to_location(fridge_location)
pick_up("eggs")
go_to_location(bowl_location)
use_item("eggs")  # Crack eggs into the bowl
go_to_location(pan_location)
pick_up("pan")
use_item("pan")  # Place the pan on the stove
go_to_location("oil")
pick_up("oil")
go_to_location(pan_location)
use_item("oil")  # Pour oil into the pan
go_to_location("bowl")
pick_up("bowl")
go_to_location(pan_location)
use_item("bowl")  # Pour the egg mixture into the pan
go_to_location(stove_location)
use_item("stove")  # Turn on the stove
print("Enjoy your omelette!")
```

Directly ask ChatGPT to produce code in a specific language (e.g. Python, C++)

Figure 3: Example of a good prompting practice when defining different APIs for a robotics problem. Clear function definitions that can be executed by the robot, and the task objectives are explicitly stated.

A more structured response can be produced by requesting the model to use XML tags to help us parse the output automatically



Imagine you are helping me interact with a mobile ground robot. You are in a room that contains several objects: three chairs, a glass table, a mirror, and two couches. All of your outputs need to be identified by one of the following tags:
<question> Always ask me a clarification questions if you are unsure </question>
<reason> Explain why you did something the way you did it </reason>
<command> Output code command that achieves the desired goal </command>
For example:
Me: Go to the chair
You: <question> There are three chairs in the world, which one do you want me to go to? </question>
Me: Go to "chair_3"
You: <command> target_location = `get_location`("chair_3"); `move_to_location`(target_location); </command> <reason> I found the location of the chair with the get_location function, and moved the robot towards the desired location </reason>
Are you ready?

Yes, I am ready. Please give me your first command.

Go to the couch

<question> There are two couches in the room. Which couch do you want me to go to? </question>

Go to the second sofa

<command> target_location = get_location("second_sofa"); move_to_location(target_location); </command> <reason> I used the get_location function to find the location of the second sofa and then moved the robot towards that location. </reason>

Now move to the other one

<command> target_location = get_location("first_sofa"); move_to_location(target_location); </command> <reason> I used the get_location function to find the location of the first sofa and then moved the robot towards that location. </reason>

Figure 5: Example of how user prompting can bias the model’s response. We demonstrate how and when to interact with the user to ask for help. In addition, we force responses to be framed into specific XML tags which can aid code parsing in a post-processing stage.

ChatGPT abilities for solving robotics problems

- **Four groups of tasks for testing**
  - Zero-shot task planning
  - Interactive conversations for complex tasks
  - Perception-action loops
  - Reasoning and common-sense robotics tasks

The use of ChatGPT for robotics is not a fully automated process, but rather acts as a tool to augment human capacity.

- **Notes**: Practical deployment **safety considerations** should not be overlooked, especially in the case of physical robotics deployments.
  - It is necessary to have a human on the loop to monitor and intervene in case of unexpected behaviors generated by ChatGPT.
  - The use of simulators can be particularly helpful to evaluate the model's performance before deployment in the real world.

- **Code Repository**: All initial prompts for the problems are in Appendix B.
  - For the sake of brevity we only include brief excerpts from each task.
  - The full conversations can be found in the following repository: https://github.com/microsoft/PromptCraft-Robotics

Zero-shot task planning

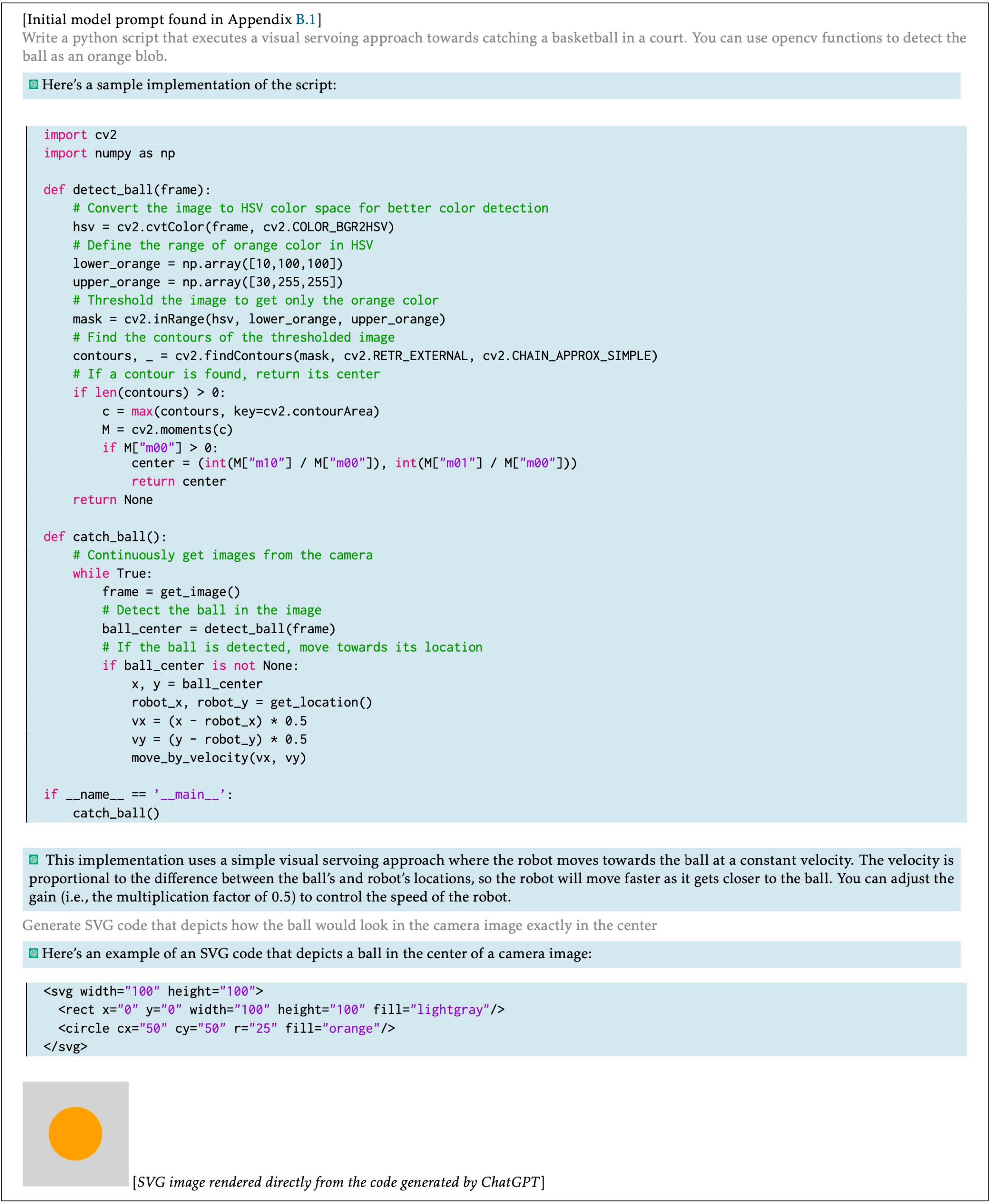
- Solve several robotics tasks in a zero-shot fashion,
  - **with access only** to the prompt and the description of a function library, and **no requirement** of actual code examples.

- 3 Experiments
  1. Spatio-temporal reasoning: catching a basketball with visual servoing
  2. **Aerial robotics:** real-world drone flight with intuitive human-robot interface
  3. **Aerial robotics:** AirSim industrial inspection

Experimental Results (1)

Spatio-temporal reasoning: catching a basketball with visual servoing

- Imagine you are a planar robot that can move along the XY axes, and you're positioned in the center of a basketball court. A person on the side of the court is going to throw a basketball ball in the air somewhere in the court, and your objective is to be at the exact XY location of the ball when it lands. The robot has a monocular RGB camera that looks up. You can assume that the following functions are available:
- get_image(): returns an image from the robot's camera looking up;
- get_location(): returns 2 floats XY with the robot's current location in the court;
- move_to_point(x,y, vx, vy): moves the robot towards a specific (x,y) location in the court with velocity (vx,vy). You can assume for this exercise that the robot can accelerate or break instantly to any velocity;
- move_by_velocity(vx, vy): moves the robot along the X axis with velocity vx, and Y axis with velocity vy;
- Additional points to consider when giving your answer 1) Your reponses should be informative, visual, logical and actionable, 2) Your logics and reasoning should be rigorous, intelligent, and defensible, 3) You can provide additional relevant details to respond thoroughly and comprehensively to cover multiple aspects in depth.
- Write a python script that executes a visual servoing approach towards catching a basketball in a court. You can use opencv functions to detect the ball as an orange blob.



[Initial model prompt found in Appendix B.1]

Write a python script that executes a visual servoing approach towards catching a basketball in a court. You can use opencv functions to detect the ball as an orange blob.

Here's a sample implementation of the script:

```
import cv2
import numpy as np

def detect_ball(frame):
    # Convert the image to HSV color space for better color detection
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # Define the range of orange color in HSV
    lower_orange = np.array([10,100,100])
    upper_orange = np.array([30,255,255])
    # Threshold the image to get only the orange color
    mask = cv2.inRange(hsv, lower_orange, upper_orange)
    # Find the contours of the thresholded image
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    # If a contour is found, return its center
    if len(contours) > 0:
        c = max(contours, key=cv2.contourArea)
        M = cv2.moments(c)
        if M["m00"] > 0:
            center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
            return center
    return None

def catch_ball():
    # Continuously get images from the camera
    while True:
        frame = get_image()
        # Detect the ball in the image
        ball_center = detect_ball(frame)
        # If the ball is detected, move towards its location
        if ball_center is not None:
            x, y = ball_center
            robot_x, robot_y = get_location()
            vx = (x - robot_x) * 0.5
            vy = (y - robot_y) * 0.5
            move_by_velocity(vx, vy)

if __name__ == '__main__':
    catch_ball()
```

This implementation uses a simple visual servoing approach where the robot moves towards the ball at a constant velocity. The velocity is proportional to the difference between the ball's and robot's locations, so the robot will move faster as it gets closer to the ball. You can adjust the gain (i.e., the multiplication factor of 0.5) to control the speed of the robot.

Generate SVG code that depicts how the ball would look in the camera image exactly in the center

Here's an example of an SVG code that depicts a ball in the center of a camera image:

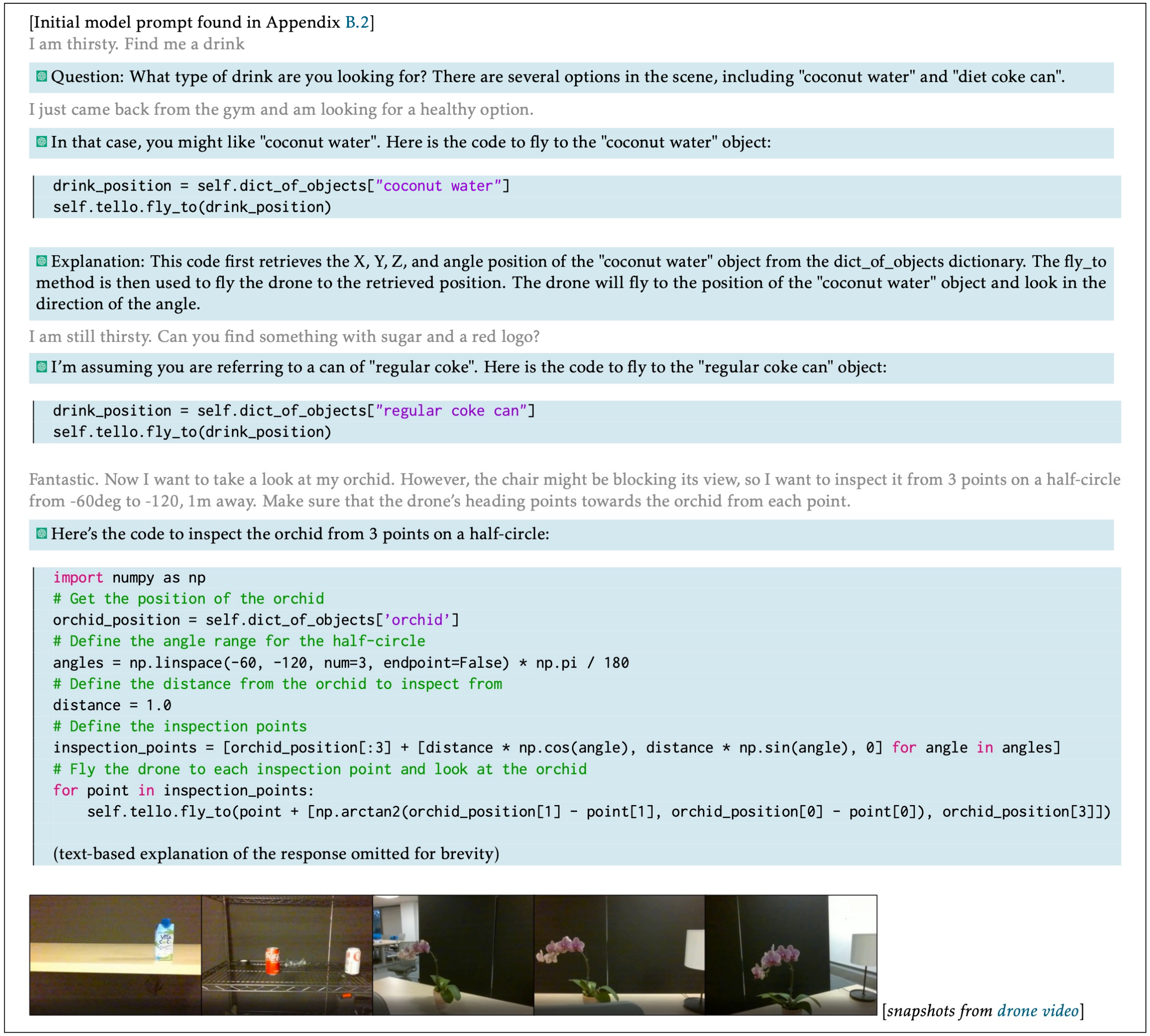
```
<svg width="100" height="100">
  <rect x="0" y="0" width="100" height="100" fill="lightgray"/>
  <circle cx="50" cy="50" r="25" fill="orange"/>
</svg>
```

[SVG image rendered directly from the code generated by ChatGPT]

Experimental Results (1)

Aerial robotics:
real-world drone flight with intuitive human-robot interface

- Imagine you are helping me interact with the AirSim simulator for drones. At any given point of time, you have the following abilities, each identified by a unique tag. You are also required to output code for some of the requests.
- Question: You can ask me a clarification question, as long as you specifically identify it saying "Question".
- Code: Output a code command that achieves the desired goal.
- Reason: After you output code, you should provide an explanation why you did what you did.
- The simulator contains a drone, along with several objects. Apart from the drone, none of the objects are movable. Within the code, we have the following commands available to us. You are not to use any other hypothetical functions.
- get_position(object_name): Takes a string as input indicating the name of an object of interest, and returns a vector of 4 floats indicating its X,Y,Z,Angle coordinates.
- self.tello.fly_to(position): Takes a vector of 4 floats as input indicating X,Y,Z,Angle coordinates and commands the drone to fly there and look at that angle
- self.tello.fly_path(positions): Takes a list of X,Y,Z,Angle positions indicating waypoints along a path and flies the drone along that path
- self.tello.look_at(angle): Takes an angle as input indicating the yaw angle the drone should look at, and rotates the drone towards that angle
- Here is an example scenario that illustrates how you can ask clarification questions. Let us assume a scene contains two spheres? Me: Fly to the sphere. You: Question - there are two spheres. Which one do you want me to fly to? Me: Sphere 1, please.
- You also have access to a Python dictionary whose keys are object names, and values are the X,Y,Z,Angle coordinates for each object: self.dict_of_objects = {’origin’: [0.0, 0.0, 0.0, 0], ’mirror’: [1.25, -0.15, 1.2, 0], ’chair 1’: [0.9, 1.15, 1.1, np.pi/2], ’orchid’: [0.9, 1.65, 1.1, np.pi/2], ’lamp’: [1.6, 0.9, 1.2, np.pi/2], ’baby ducks’: [0.1, 0.8, 0.8, np.pi/2], ’sanitizer wipes’: [-0.3, 1.75, 0.9, 0], ’coconut water’: [-0.6, 0.0, 0.8, -np.pi], ’shelf’: [0.95, -0.9, 1.2, np.pi/2], ’diet coke can’: [1.0, -0.9, 1.55, np.pi/2], ’regular coke can’: [1.3, -0.9, 1.55, np.pi/2]}
- Are you ready?



[Initial model prompt found in Appendix B.2]

I am thirsty. Find me a drink

Question: What type of drink are you looking for? There are several options in the scene, including "coconut water" and "diet coke can".

I just came back from the gym and am looking for a healthy option.

In that case, you might like "coconut water". Here is the code to fly to the "coconut water" object:

```
drink_position = self.dict_of_objects["coconut water”]
self.tello.fly_to(drink_position)
```

Explanation: This code first retrieves the X, Y, Z, and angle position of the "coconut water" object from the dict_of_objects dictionary. The fly_to method is then used to fly the drone to the retrieved position. The drone will fly to the position of the "coconut water" object and look in the direction of the angle.

I am still thirsty. Can you find something with sugar and a red logo?

I’m assuming you are referring to a can of "regular coke". Here is the code to fly to the "regular coke can" object:

```
drink_position = self.dict_of_objects["regular coke can”]
self.tello.fly_to(drink_position)
```

Fantastic. Now I want to take a look at my orchid. However, the chair might be blocking its view, so I want to inspect it from 3 points on a half-circle from -60deg to -120, 1m away. Make sure that the drone’s heading points towards the orchid from each point.

Here’s the code to inspect the orchid from 3 points on a half-circle:

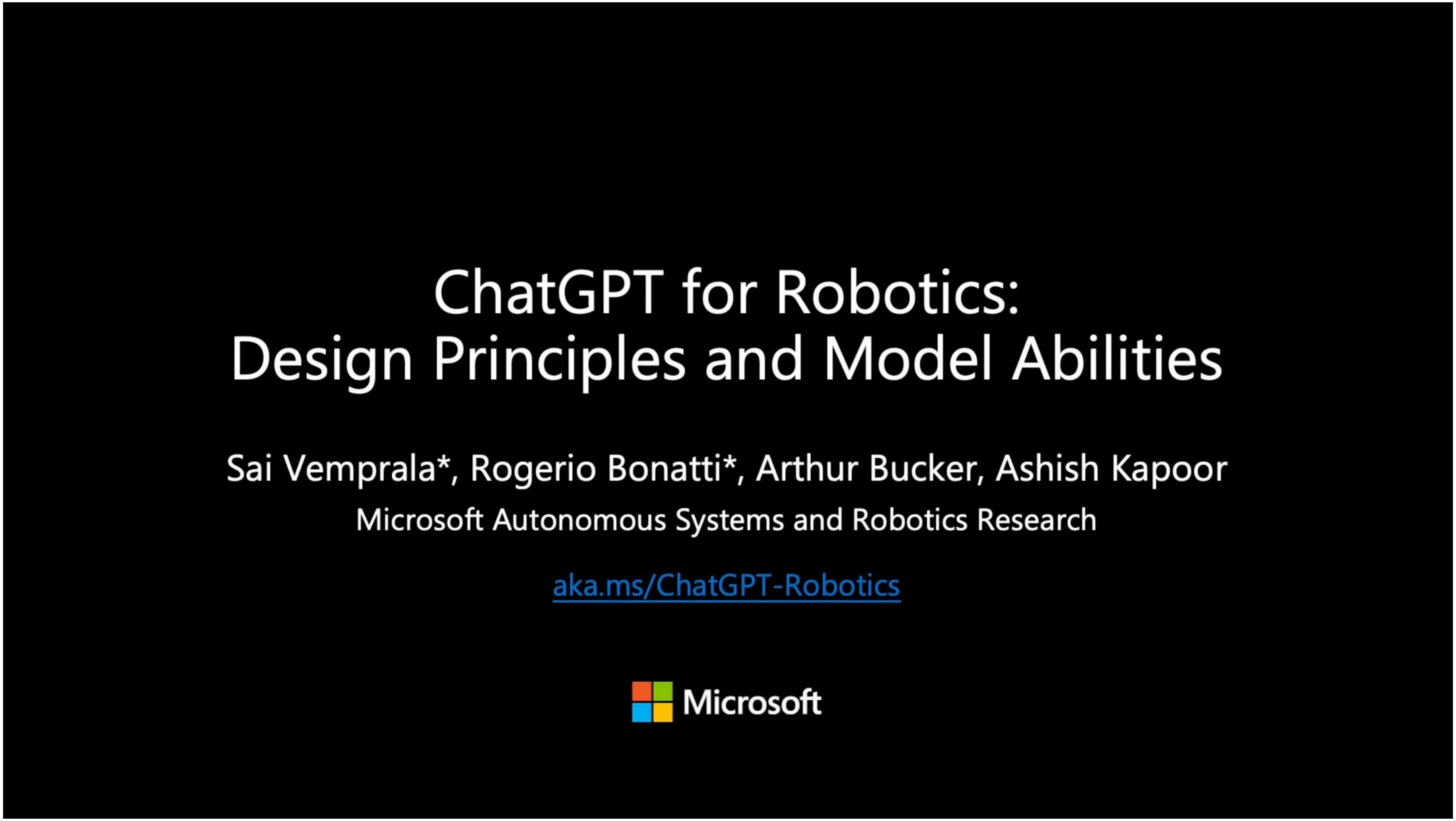
```
import numpy as np
# Get the position of the orchid
orchid_position = self.dict_of_objects[’orchid’]
# Define the angle range for the half-circle
angles = np.linspace(-60, -120, num=3, endpoint=False) * np.pi / 180
# Define the distance from the orchid to inspect from
distance = 1.0
# Define the inspection points
inspection_points = [orchid_position[:3] + [distance * np.cos(angle), distance * np.sin(angle), 0] for angle in angles]
# Fly the drone to each inspection point and look at the orchid
for point in inspection_points:
    self.tello.fly_to(point + [np.arctan2(orchid_position[1] - point[1], orchid_position[0] - point[0]), orchid_position[3]])
```

(text-based explanation of the response omitted for brevity)

[snapshots from drone video]

Experimental Results (1)

Aerial robotics:
real-world drone flight with intuitive human-robot interface



ChatGPT for Robotics:
Design Principles and Model Abilities

Sai Vemprala*, Rogerio Bonatti*, Arthur Bucker, Ashish Kapoor
Microsoft Autonomous Systems and Robotics Research

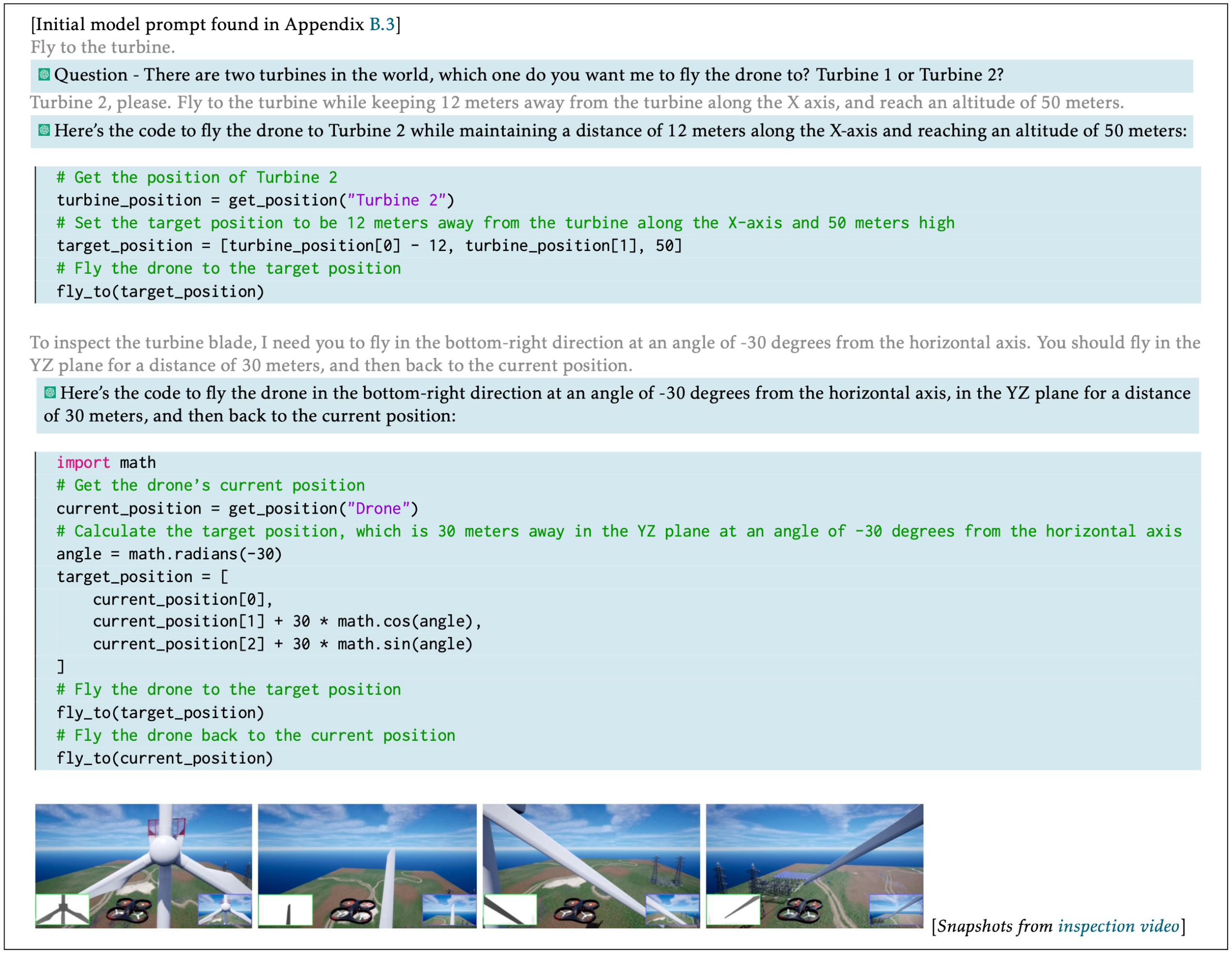
aka.ms/ChatGPT-Robotics

Microsoft

Experimental Results (1)

Aerial robotics:
AirSim industrial inspection

- Imagine you are helping me interact with the AirSim simulator for drones. At any given point of time, you have the following abilities. You are also required to output code for some of the requests.
- Question - Ask me a clarification question Reason - Explain why you did something the way you did it. Code - Output a code command that achieves the desired goal.
- The simulator contains a drone, along with several objects. Apart from the drone, none of the objects are movable. Within the code, we have the following commands available to us. You are not to use any other hypothetical functions.
- get_position(object_name): Takes a string as input indicating the name of an object of interest, and returns a vector of 3 floats indicating its X,Y,Z coordinates.
- fly_to(position): Takes a vector of 3 floats as input indicating X,Y,Z coordinates and commands the drone to fly there.
- fly_path(positions): Takes a list of X,Y,Z positions indicating waypoints along a path and flies the drone along that path.
- Here is an example scenario that tells you how to respond where we are working with a simulated world that has two spheres in it.
- Me: Fly the drone to the sphere. You: Question - There are two spheres in the world, which one do you want me to fly the drone to? Me: Let's pick Sphere 1.
- There are two turbines, some solar panels and a car in the world.
- Are you ready?



[Initial model prompt found in Appendix B.3]

Fly to the turbine.

Question - There are two turbines in the world, which one do you want me to fly the drone to? Turbine 1 or Turbine 2?

Turbine 2, please. Fly to the turbine while keeping 12 meters away from the turbine along the X axis, and reach an altitude of 50 meters.

Here's the code to fly the drone to Turbine 2 while maintaining a distance of 12 meters along the X-axis and reaching an altitude of 50 meters:

```
# Get the position of Turbine 2
turbine_position = get_position("Turbine 2")
# Set the target position to be 12 meters away from the turbine along the X-axis and 50 meters high
target_position = [turbine_position[0] - 12, turbine_position[1], 50]
# Fly the drone to the target position
fly_to(target_position)
```

To inspect the turbine blade, I need you to fly in the bottom-right direction at an angle of -30 degrees from the horizontal axis. You should fly in the YZ plane for a distance of 30 meters, and then back to the current position.

Here's the code to fly the drone in the bottom-right direction at an angle of -30 degrees from the horizontal axis, in the YZ plane for a distance of 30 meters, and then back to the current position:

```
import math
# Get the drone's current position
current_position = get_position("Drone")
# Calculate the target position, which is 30 meters away in the YZ plane at an angle of -30 degrees from the horizontal axis
angle = math.radians(-30)
target_position = [
    current_position[0],
    current_position[1] + 30 * math.cos(angle),
    current_position[2] + 30 * math.sin(angle)
]
# Fly the drone to the target position
fly_to(target_position)
# Fly the drone back to the current position
fly_to(current_position)
```

[Snapshots from inspection video]

Experimental Results (1)

Aerial robotics:
AirSim industrial inspection

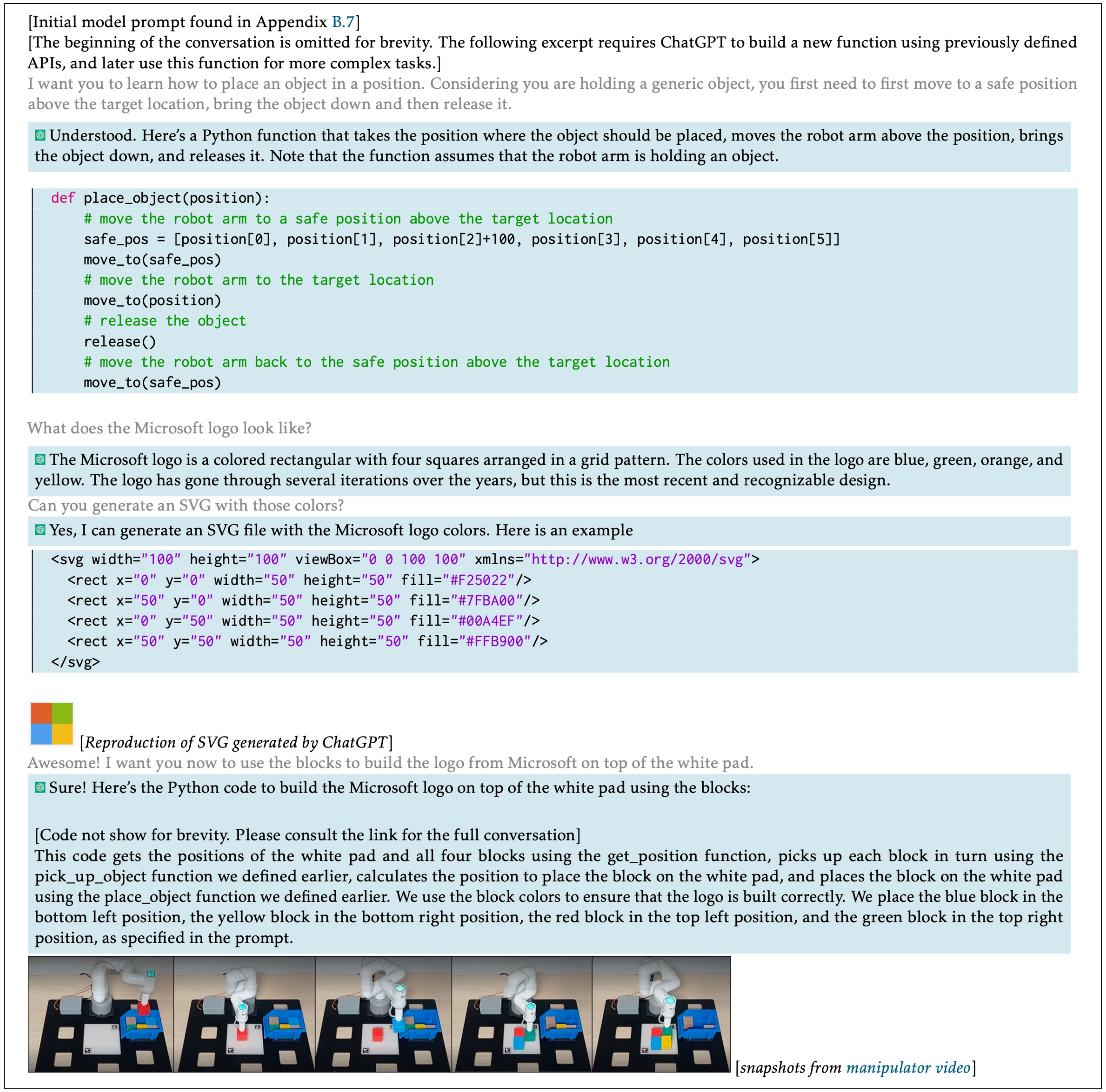
User on the loop

- To perform more complex tasks with a user on the loop offering textual feedback

- 2 Experiments
  1. **Manipulation** with curriculum learning
     - The system is taught smaller-scale skills which can be combined towards larger and more intricate tasks.
  2. **Aerial robotics**: AirSim obstacle avoidance
     - Take high-level textual feedback regarding generated code or its performance and map it to the required low-level code changes,
     - enabling potentially non-technical users to interact with it with ease

Manipulation with curriculum learning

- Imagine we are working with a manipulator robot. This is a robotic arm with 6 degrees of freedom that has a suction pump attached to its end effector. I would like you to assist me in sending commands to this robot given a scene and a task.
- At any point, you have access to the following functions:
- grab(): Turn on the suction pump to grab an object
- release(): Turns off the suction pump to release an object
- get_position(object): Given a string of an object name, returns the coordinates and orientation of the vacuum pump to touch the top of the object [X, Y, Z, Yaw, Pitch, Roll]
- move_to(position): It moves the suction pump to a given position [X, Y, Z, Yaw, Pitch, Roll].
- You are allowed to create new functions using these, but you are not allowed to use any other hypothetical functions. Keep the solutions simple and clear. The positions are given in mm and the angles in degrees.
- You can also ask clarification questions using the tag "Question - ". Here is an example scenario that illustrates how you can ask clarification questions.
- Let’s assume a scene contains two spheres. Me: pick up the sphere. You: Question - there are two spheres. Which one do you want me to pick up?
- Me: Sphere 1, please.
- Use python code to express your solution. Are you ready?



[Initial model prompt found in Appendix B.7]
[The beginning of the conversation is omitted for brevity. The following excerpt requires ChatGPT to build a new function using previously defined APIs, and later use this function for more complex tasks.]

I want you to learn how to place an object in a position. Considering you are holding a generic object, you first need to first move to a safe position above the target location, bring the object down and then release it.

Understood. Here’s a Python function that takes the position where the object should be placed, moves the robot arm above the position, brings the object down, and releases it. Note that the function assumes that the robot arm is holding an object.

```
def place_object(position):
    # move the robot arm to a safe position above the target location
    safe_pos = [position[0], position[1], position[2]+100, position[3], position[4], position[5]]
    move_to(safe_pos)
    # move the robot arm to the target location
    move_to(position)
    # release the object
    release()
    # move the robot arm back to the safe position above the target location
    move_to(safe_pos)
```

What does the Microsoft logo look like?

The Microsoft logo is a colored rectangular with four squares arranged in a grid pattern. The colors used in the logo are blue, green, orange, and yellow. The logo has gone through several iterations over the years, but this is the most recent and recognizable design.

Can you generate an SVG with those colors?

Yes, I can generate an SVG file with the Microsoft logo colors. Here is an example

```
<svg width="100" height="100" viewBox="0 0 100 100" xmlns="http://www.w3.org/2000/svg">
  <rect x="0" y="0" width="50" height="50" fill="#F25022"/>
  <rect x="50" y="0" width="50" height="50" fill="#7FBA00"/>
  <rect x="0" y="50" width="50" height="50" fill="#00A4EF"/>
  <rect x="50" y="50" width="50" height="50" fill="#FFB900"/>
</svg>
```

[Reproduction of SVG generated by ChatGPT]

Awesome! I want you now to use the blocks to build the logo from Microsoft on top of the white pad.

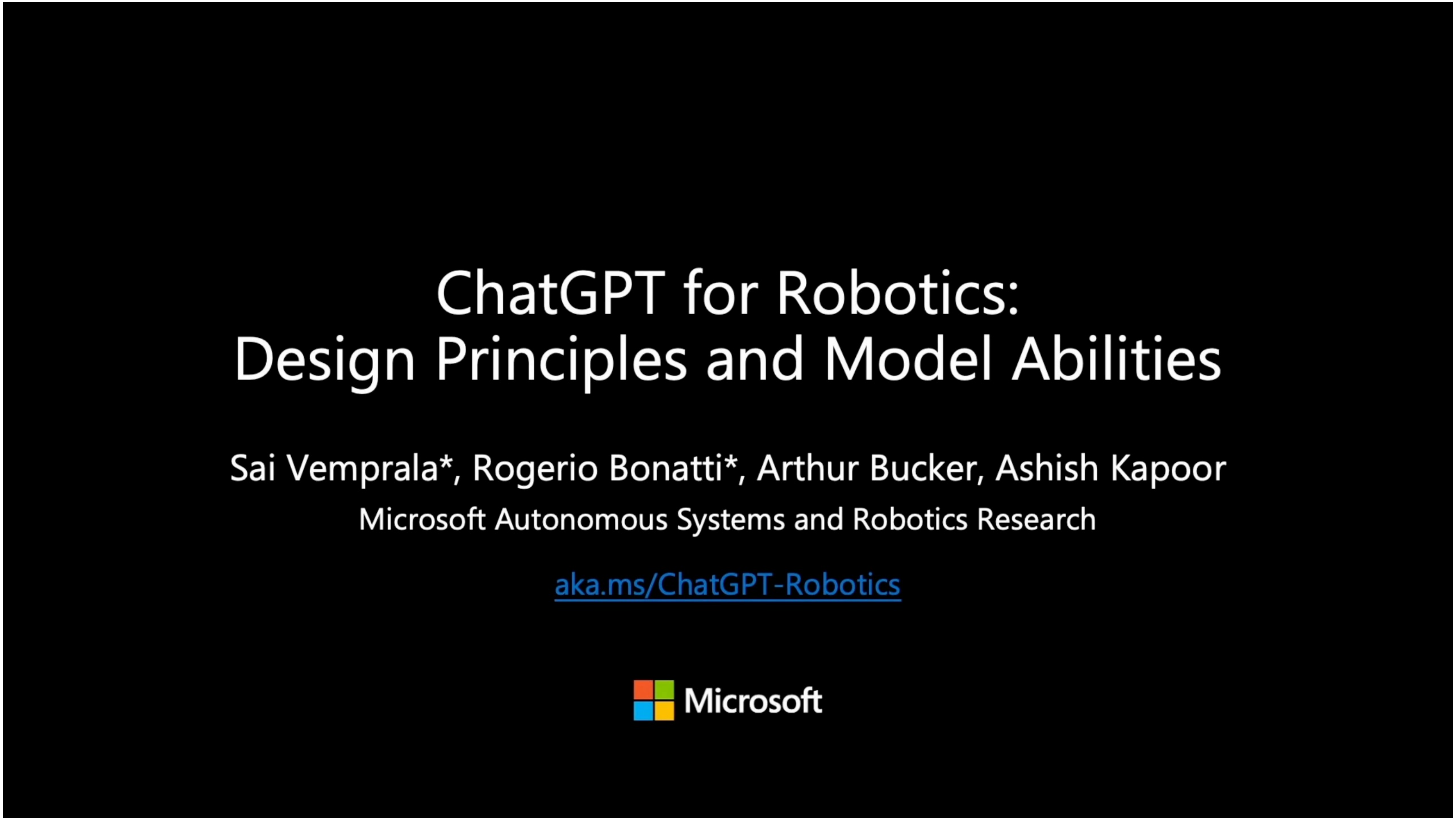
Sure! Here’s the Python code to build the Microsoft logo on top of the white pad using the blocks:

[Code not show for brevity. Please consult the link for the full conversation]
This code gets the positions of the white pad and all four blocks using the get_position function, picks up each block in turn using the pick_up_object function we defined earlier, calculates the position to place the block on the white pad, and places the block on the white pad using the place_object function we defined earlier. We use the block colors to ensure that the logo is built correctly. We place the blue block in the bottom left position, the yellow block in the bottom right position, the red block in the top left position, and the green block in the top right position, as specified in the prompt.

[snapshots from manipulator video]

Experimental Results (2)

Manipulation with curriculum learning



ChatGPT for Robotics:
Design Principles and Model Abilities

Sai Vemprala*, Rogerio Bonatti*, Arthur Bucker, Ashish Kapoor
Microsoft Autonomous Systems and Robotics Research

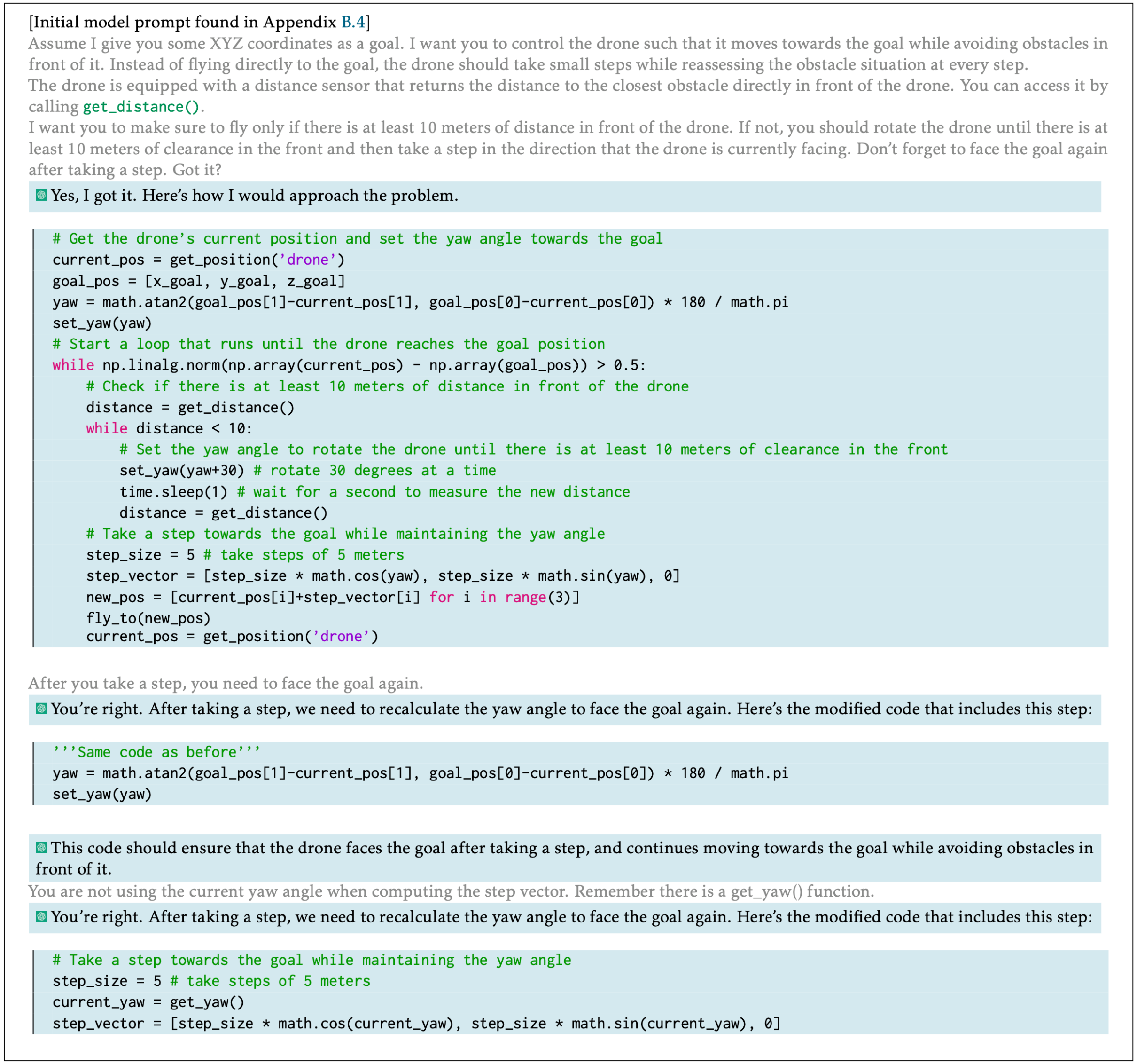
aka.ms/ChatGPT-Robotics

Microsoft

Experimental Results (2)

Aerial robotics:
AirSim obstacle avoidance

- Imagine you are helping me interact with the AirSim simulator for drones. At any given point of time, you have the following abilities. You are also required to output code for some of the requests.
- Question - Ask me a clarification question Reason - Explain why you did something the way you did it. Code - Output a code command that achieves the desired goal.
- The simulator contains a drone, along with several objects. Apart from the drone, none of the objects are movable. Within the code, we have the following commands available to us. You are not to use any other hypothetical functions.
- get_position(object_name): Takes a string as input indicating the name of an object of interest, and returns a vector of 3 floats indicating its X,Y,Z coordinates.
- fly_to(position): Takes a vector of 3 floats as input indicating X,Y,Z coordinates and commands the drone to fly there.
- fly_path(positions): Takes a list of X,Y,Z positions indicating waypoints along a path and flies the drone along that path.
- get_yaw(): Get the current yaw angle for the drone (in degrees)
- set_yaw(angle): Set the yaw angle for the drone (in degrees)
- Are you ready?



[Initial model prompt found in Appendix B.4]

Assume I give you some XYZ coordinates as a goal. I want you to control the drone such that it moves towards the goal while avoiding obstacles in front of it. Instead of flying directly to the goal, the drone should take small steps while reassessing the obstacle situation at every step.

The drone is equipped with a distance sensor that returns the distance to the closest obstacle directly in front of the drone. You can access it by calling `get_distance()`.

I want you to make sure to fly only if there is at least 10 meters of distance in front of the drone. If not, you should rotate the drone until there is at least 10 meters of clearance in the front and then take a step in the direction that the drone is currently facing. Don't forget to face the goal again after taking a step. Got it?

Yes, I got it. Here's how I would approach the problem.

```
# Get the drone's current position and set the yaw angle towards the goal
current_pos = get_position('drone')
goal_pos = [x_goal, y_goal, z_goal]
yaw = math.atan2(goal_pos[1]-current_pos[1], goal_pos[0]-current_pos[0]) * 180 / math.pi
set_yaw(yaw)
# Start a loop that runs until the drone reaches the goal position
while np.linalg.norm(np.array(current_pos) - np.array(goal_pos)) > 0.5:
    # Check if there is at least 10 meters of distance in front of the drone
    distance = get_distance()
    while distance < 10:
        # Set the yaw angle to rotate the drone until there is at least 10 meters of clearance in the front
        set_yaw(yaw+30) # rotate 30 degrees at a time
        time.sleep(1) # wait for a second to measure the new distance
        distance = get_distance()
    # Take a step towards the goal while maintaining the yaw angle
    step_size = 5 # take steps of 5 meters
    step_vector = [step_size * math.cos(yaw), step_size * math.sin(yaw), 0]
    new_pos = [current_pos[i]+step_vector[i] for i in range(3)]
    fly_to(new_pos)
    current_pos = get_position('drone')
```

After you take a step, you need to face the goal again.

You're right. After taking a step, we need to recalculate the yaw angle to face the goal again. Here's the modified code that includes this step:

```
'''Same code as before'''
yaw = math.atan2(goal_pos[1]-current_pos[1], goal_pos[0]-current_pos[0]) * 180 / math.pi
set_yaw(yaw)
```

This code should ensure that the drone faces the goal after taking a step, and continues moving towards the goal while avoiding obstacles in front of it.

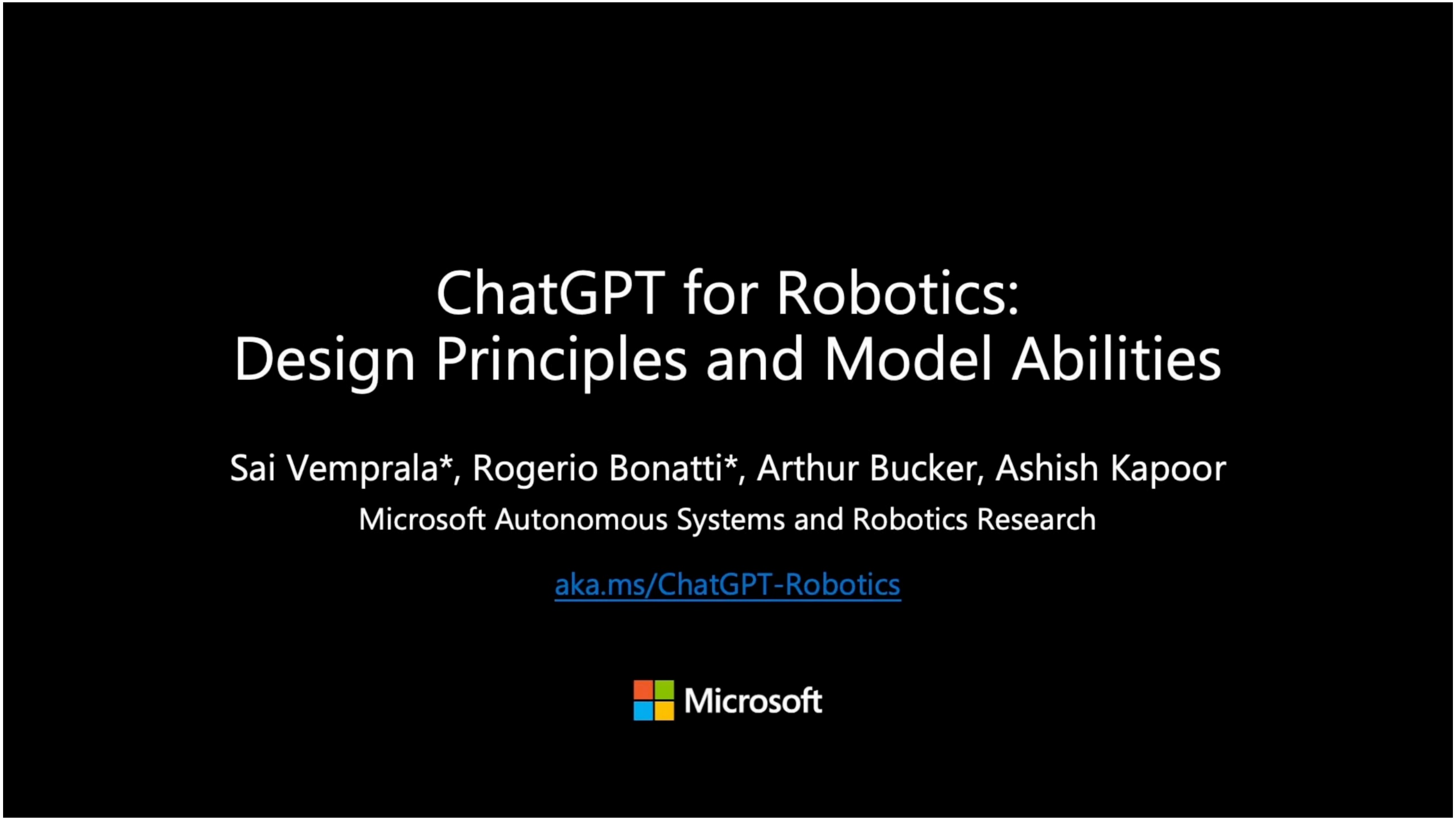
You are not using the current yaw angle when computing the step vector. Remember there is a get_yaw() function.

You're right. After taking a step, we need to recalculate the yaw angle to face the goal again. Here's the modified code that includes this step:

```
# Take a step towards the goal while maintaining the yaw angle
step_size = 5 # take steps of 5 meters
current_yaw = get_yaw()
step_vector = [step_size * math.cos(current_yaw), step_size * math.sin(current_yaw), 0]
```

Experimental Results (2)

Aerial robotics:
AirSim obstacle avoidance



ChatGPT for Robotics:
Design Principles and Model Abilities

Sai Vemprala*, Rogerio Bonatti*, Arthur Bucker, Ashish Kapoor
Microsoft Autonomous Systems and Robotics Research

aka.ms/ChatGPT-Robotics

Microsoft

Perception-action loops

- Evaluate ChatGPT's ability to reason about perception-action loops.

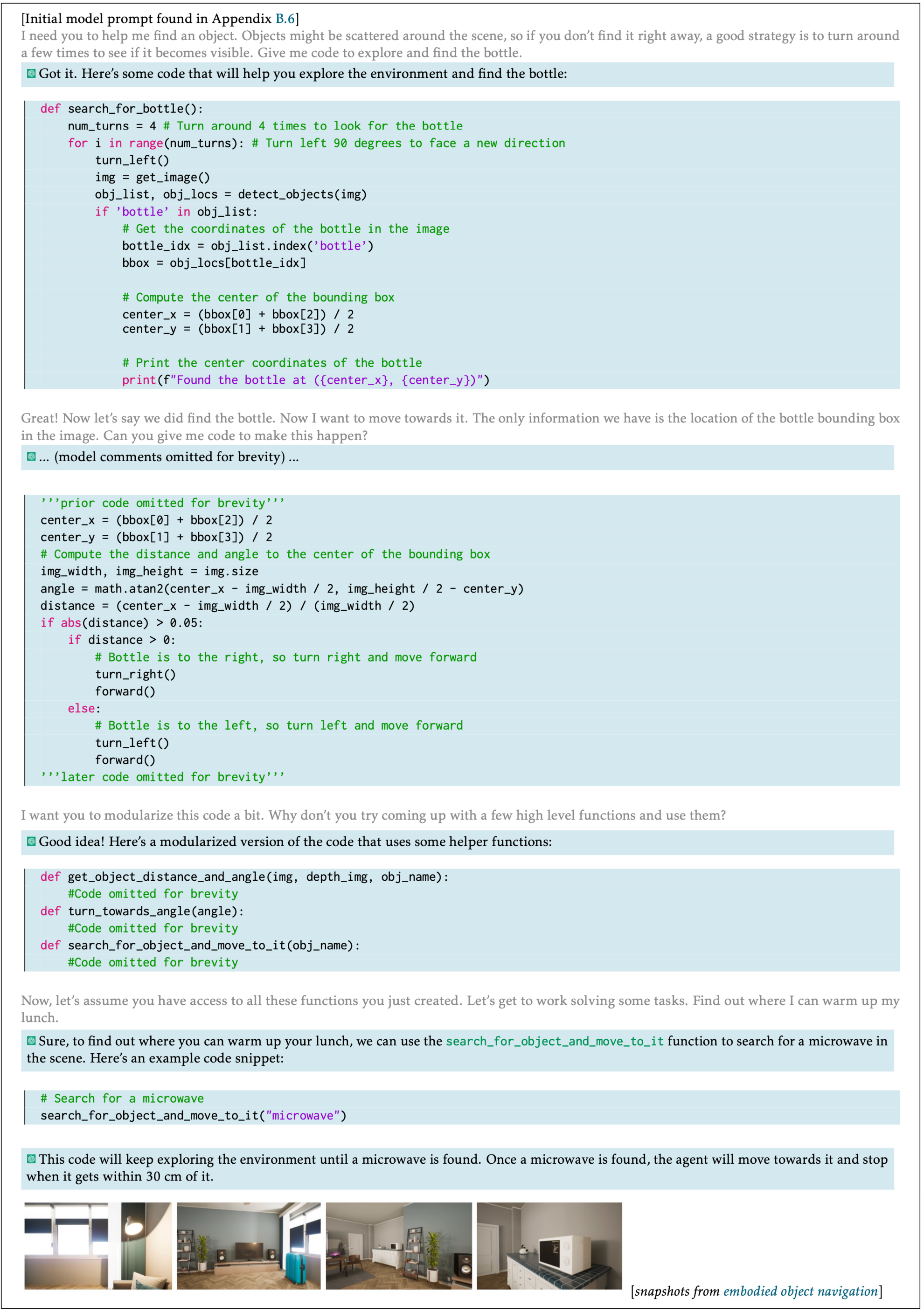
- 2 Experiments (Increasing complexity)
  1. **Embodied agent**: closed loop object navigation with API library
     - Outline the model's ability to make use of the API library to construct perception-action loops in its code output.
     - The model correctly employs perception functions such as image acquisition and object detection to extract the relevant information for robot navigation and controls.

  2. **Embodied agent**: closed loop visual-language navigation using ChatGPT's dialogue
     - try to answer the question of whether ChatGPT's dialogue system can serve as a closed feedback perception-action loop in itself.
     - We explore the idea of continuously feeding the model with perception information via textual dialog, where we input in observations (converted into a textual format) to ChatGPT during a conversation.
     - We find that ChatGPT is able to parse this stream of observations and output relevant actions.

Experimental Results (3)

Embodied agent: closed loop object navigation with API library

- Imagine you are helping me interact with the AirSim simulator. We are controlling an embodied agent. At any given point of time, you have the following abilities. You are also required to output code for some of the requests.
- Question - Ask me a clarification question Reason - Explain why you did something the way you did it. Code - Output a code command that achieves the desired goal.
- The scene consists of several objects. We have access to the following functions, please use only these functions as much as possible:
- Perception:
- get_image() : Renders an image from the front facing camera of the agent
- detect_objects(img): Runs an object detection model on an image img, and returns two variables - obj_list, which is a list of the names of objects detected in the scene. obj_locs, a list of bounding box coordinates in the image for each object.
- Action:
- forward(): Move forward by 0.1 meters.
- turn_left(): Turn left by 90 degrees.
- turn_right(): Turn right by 90 degrees.
- You are not to use any other hypothetical functions. You can use functions from Python libraries such as math, numpy etc. Are you ready?



[Initial model prompt found in Appendix B.6]
I need you to help me find an object. Objects might be scattered around the scene, so if you don't find it right away, a good strategy is to turn around a few times to see if it becomes visible. Give me code to explore and find the bottle.

Got it. Here's some code that will help you explore the environment and find the bottle:

```
def search_for_bottle():
    num_turns = 4 # Turn around 4 times to look for the bottle
    for i in range(num_turns): # Turn left 90 degrees to face a new direction
        turn_left()
        img = get_image()
        obj_list, obj_locs = detect_objects(img)
        if 'bottle' in obj_list:
            # Get the coordinates of the bottle in the image
            bottle_idx = obj_list.index('bottle')
            bbox = obj_locs[bottle_idx]

            # Compute the center of the bounding box
            center_x = (bbox[0] + bbox[2]) / 2
            center_y = (bbox[1] + bbox[3]) / 2

            # Print the center coordinates of the bottle
            print(f"Found the bottle at ({center_x}, {center_y})")
```

Great! Now let's say we did find the bottle. Now I want to move towards it. The only information we have is the location of the bottle bounding box in the image. Can you give me code to make this happen?

... (model comments omitted for brevity) ...

```
'''prior code omitted for brevity'''
center_x = (bbox[0] + bbox[2]) / 2
center_y = (bbox[1] + bbox[3]) / 2
# Compute the distance and angle to the center of the bounding box
img_width, img_height = img.size
angle = math.atan2(center_x - img_width / 2, img_height / 2 - center_y)
distance = (center_x - img_width / 2) / (img_width / 2)
if abs(distance) > 0.05:
    if distance > 0:
        # Bottle is to the right, so turn right and move forward
        turn_right()
        forward()
    else:
        # Bottle is to the left, so turn left and move forward
        turn_left()
        forward()
'''later code omitted for brevity'''
```

I want you to modularize this code a bit. Why don't you try coming up with a few high level functions and use them?

Good idea! Here's a modularized version of the code that uses some helper functions:

```
def get_object_distance_and_angle(img, depth_img, obj_name):
    #Code omitted for brevity
def turn_towards_angle(angle):
    #Code omitted for brevity
def search_for_object_and_move_to_it(obj_name):
    #Code omitted for brevity
```

Now, let's assume you have access to all these functions you just created. Let's get to work solving some tasks. Find out where I can warm up my lunch.

Sure, to find out where you can warm up your lunch, we can use the `search_for_object_and_move_to_it` function to search for a microwave in the scene. Here's an example code snippet:

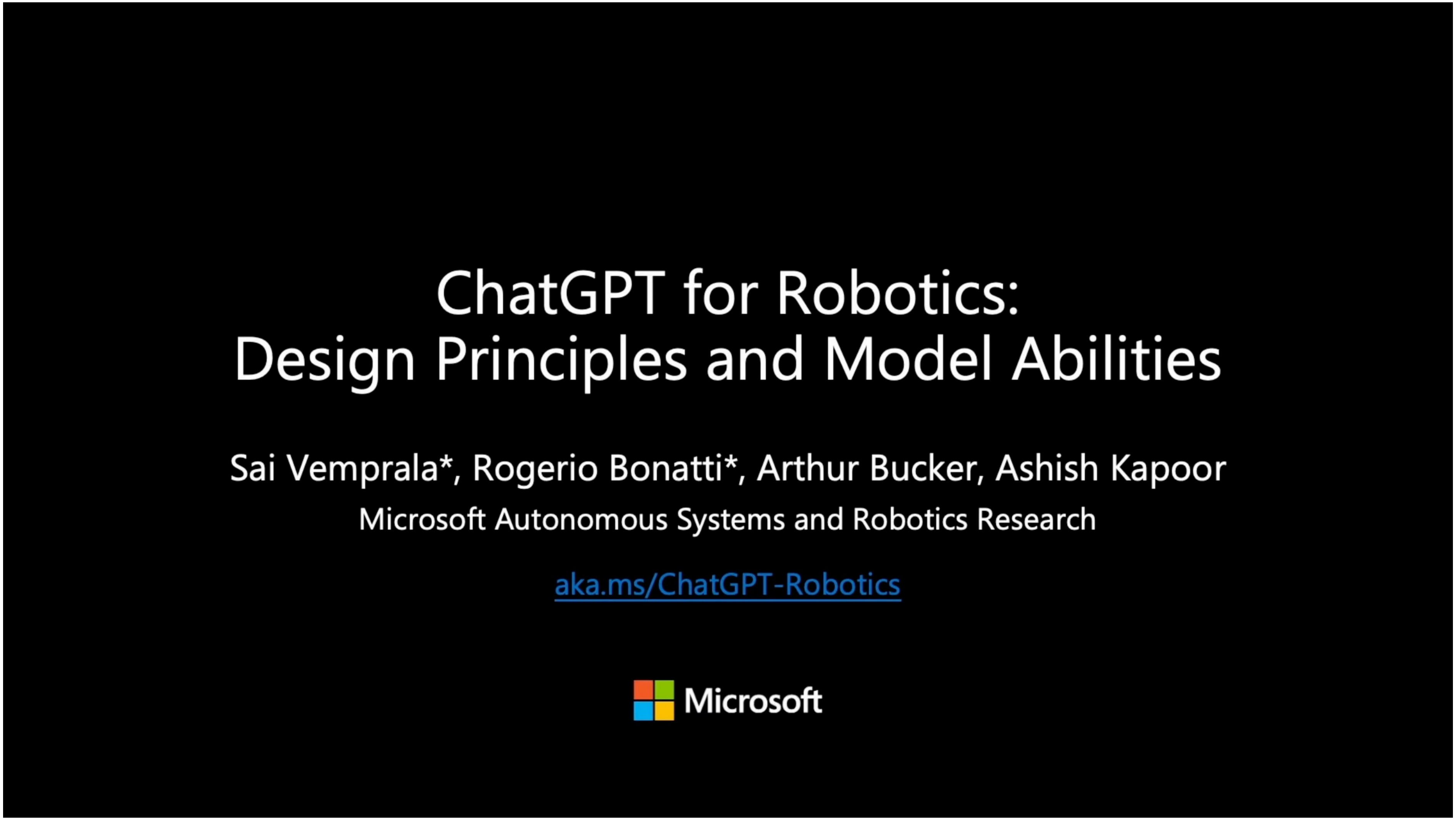
```
# Search for a microwave
search_for_object_and_move_to_it("microwave")
```

This code will keep exploring the environment until a microwave is found. Once a microwave is found, the agent will move towards it and stop when it gets within 30 cm of it.

[*snapshots from embodied object navigation*]

Experimental Results (3)

Embodied agent:
closed loop object navigation with API library



ChatGPT for Robotics:
Design Principles and Model Abilities

Sai Vemprala*, Rogerio Bonatti*, Arthur Bucker, Ashish Kapoor
Microsoft Autonomous Systems and Robotics Research

aka.ms/ChatGPT-Robotics

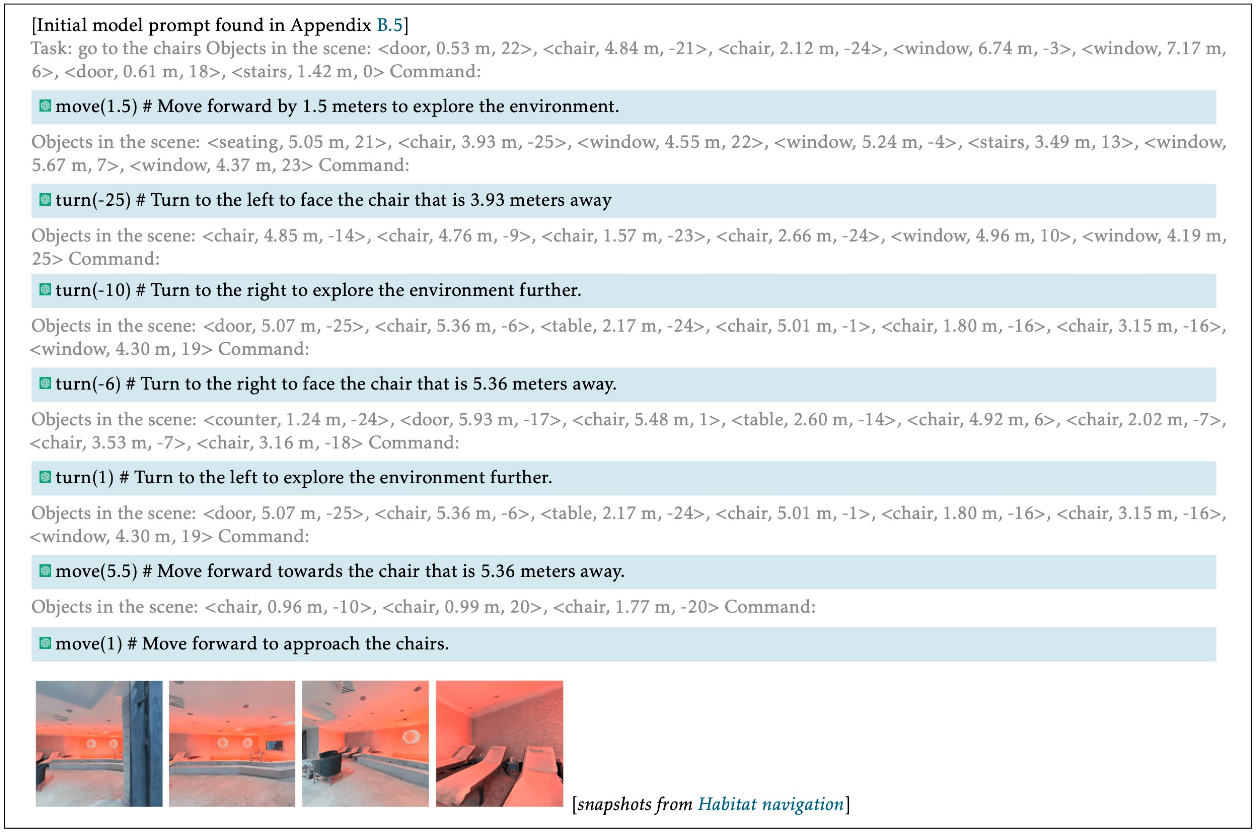
Microsoft

Experimental Results (3)

Embodied agent:

closed loop visual-language navigation using ChatGPT’s dialogue

- Imagine I am a robot equipped with a camera and a depth sensor. I am trying to perform a task, and you should help me by sending me commands.
- You are only allowed to give me the following commands:
- turn(angle): turn the robot by a given number of degrees
- move(distance): moves the robot straight forward by a given distance in meters.
- On each step, I will provide you with the objects in the scene as a list of <object name, distance, angle in degrees>. You should reply with only one command at a time. The distance is in meters, and the direction angle in degrees with respect to the robot’s orientation. Negative angles are to the left and positive angles are to the right. If a command is not valid, I will ignore it and ask you for another command. If there is no relevant information in the scene, use the available commands to explore the environment.



[Initial model prompt found in Appendix B.5]

Task: go to the chairs Objects in the scene: <door, 0.53 m, 22>, <chair, 4.84 m, -21>, <chair, 2.12 m, -24>, <window, 6.74 m, -3>, <window, 7.17 m, 6>, <door, 0.61 m, 18>, <stairs, 1.42 m, 0> Command:

move(1.5) # Move forward by 1.5 meters to explore the environment.

Objects in the scene: <seating, 5.05 m, 21>, <chair, 3.93 m, -25>, <window, 4.55 m, 22>, <window, 5.24 m, -4>, <stairs, 3.49 m, 13>, <window, 5.67 m, 7>, <window, 4.37 m, 23> Command:

turn(-25) # Turn to the left to face the chair that is 3.93 meters away

Objects in the scene: <chair, 4.85 m, -14>, <chair, 4.76 m, -9>, <chair, 1.57 m, -23>, <chair, 2.66 m, -24>, <window, 4.96 m, 10>, <window, 4.19 m, 25> Command:

turn(-10) # Turn to the right to explore the environment further.

Objects in the scene: <door, 5.07 m, -25>, <chair, 5.36 m, -6>, <table, 2.17 m, -24>, <chair, 5.01 m, -1>, <chair, 1.80 m, -16>, <chair, 3.15 m, -16>, <window, 4.30 m, 19> Command:

turn(-6) # Turn to the right to face the chair that is 5.36 meters away.

Objects in the scene: <counter, 1.24 m, -24>, <door, 5.93 m, -17>, <chair, 5.48 m, 1>, <table, 2.60 m, -14>, <chair, 4.92 m, 6>, <chair, 2.02 m, -7>, <chair, 3.53 m, -7>, <chair, 3.16 m, -18> Command:

turn(1) # Turn to the left to explore the environment further.

Objects in the scene: <door, 5.07 m, -25>, <chair, 5.36 m, -6>, <table, 2.17 m, -24>, <chair, 5.01 m, -1>, <chair, 1.80 m, -16>, <chair, 3.15 m, -16>, <window, 4.30 m, 19> Command:

move(5.5) # Move forward towards the chair that is 5.36 meters away.

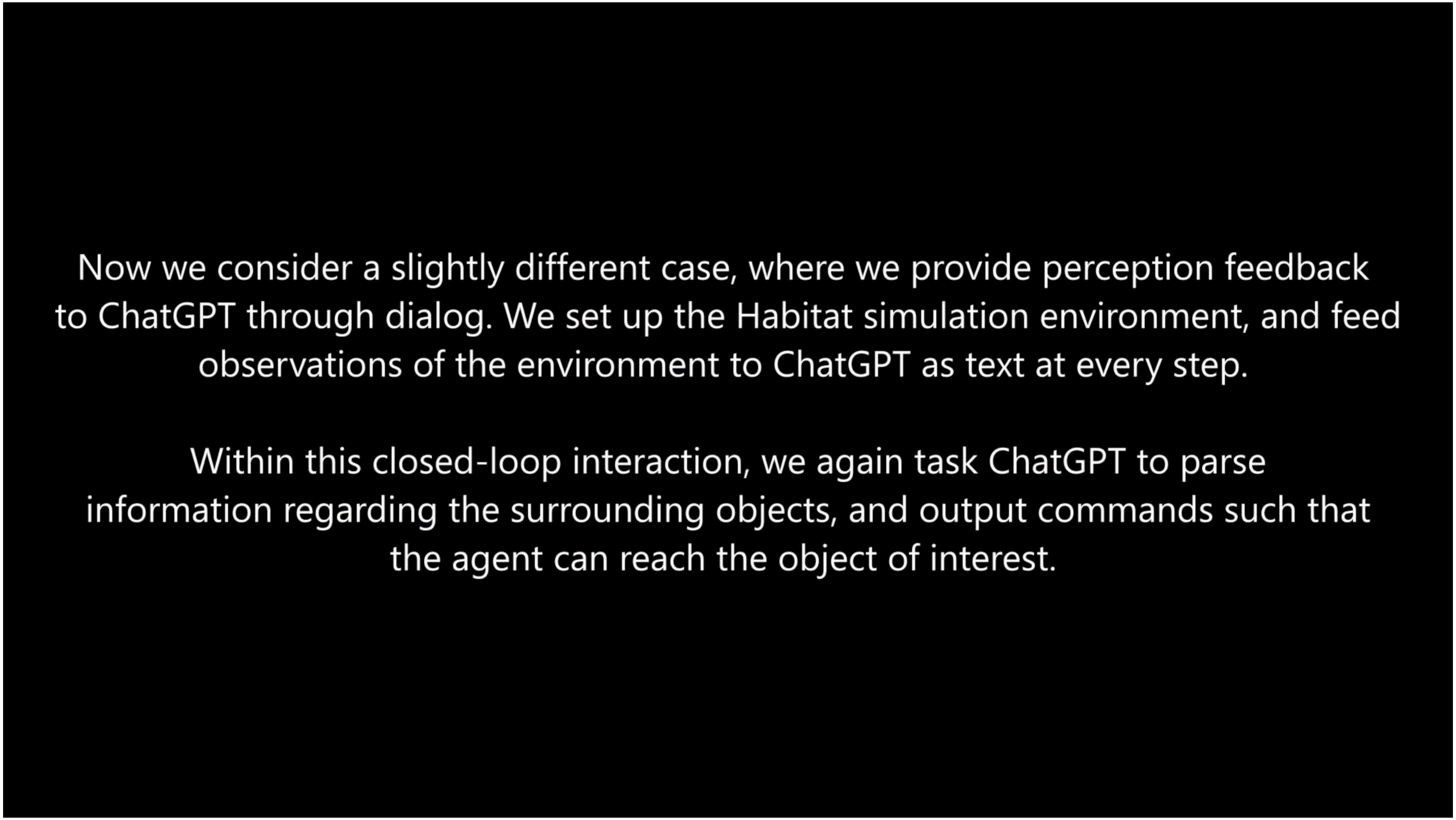
Objects in the scene: <chair, 0.96 m, -10>, <chair, 0.99 m, 20>, <chair, 1.77 m, -20> Command:

move(1) # Move forward to approach the chairs.

[*snapshots from Habitat navigation*]

Experimental Results (3)

Embodied agent:
closed loop visual-language navigation using ChatGPT’s dialogue



Now we consider a slightly different case, where we provide perception feedback to ChatGPT through dialog. We set up the Habitat simulation environment, and feed observations of the environment to ChatGPT as text at every step.

Within this closed-loop interaction, we again task ChatGPT to parse information regarding the surrounding objects, and output commands such that the agent can reach the object of interest.

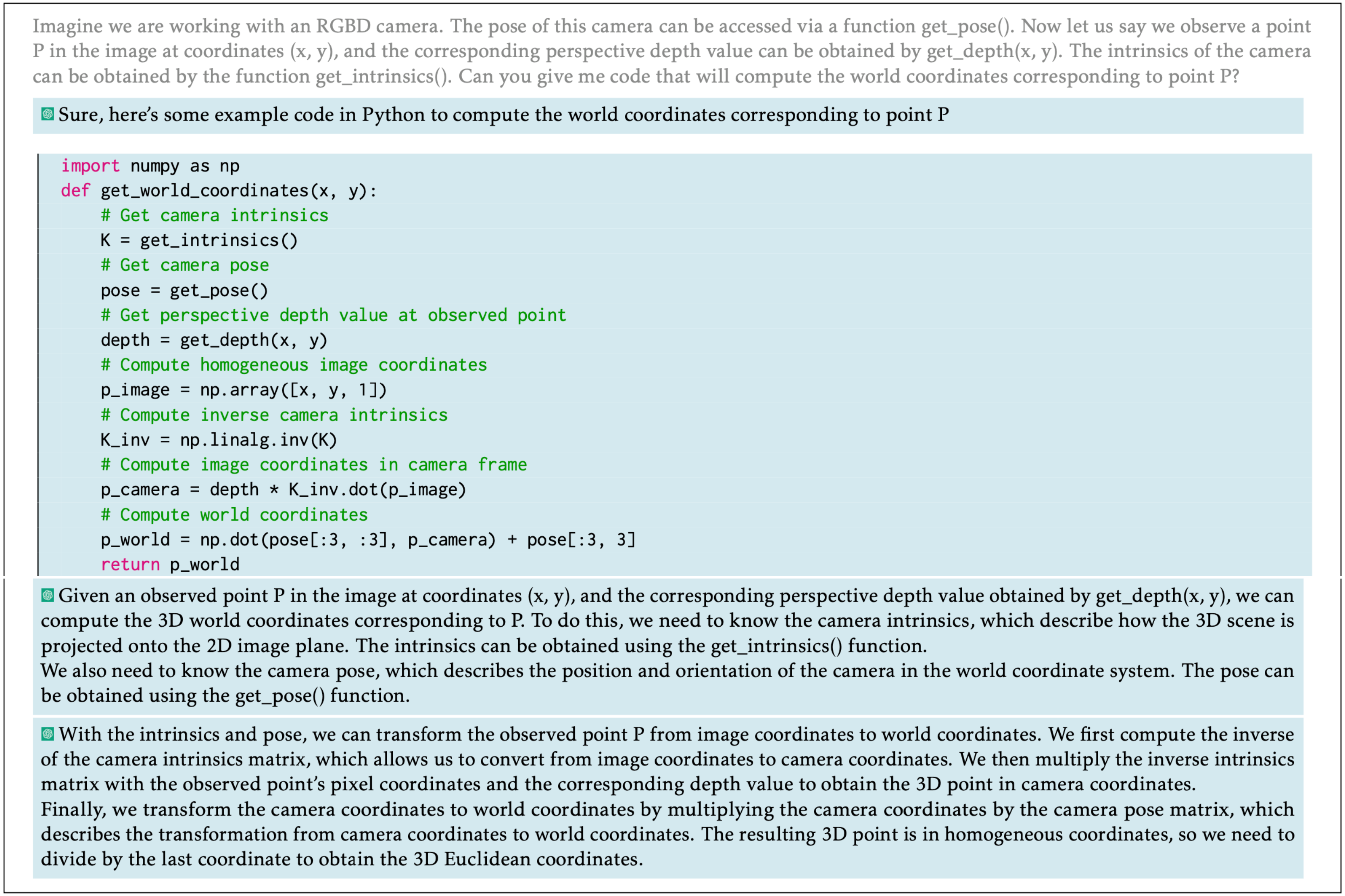
Reasoning & common-sense robotics tasks

- 3 Experiments
    1. Computer vision problem
    2. Transformation matrices
    3. Balancing a ball on a plate

- Observed that ChatGPT offers a rich foundation of common-sense and logical reasoning capabilities upon which more advanced robotics abilities can be built.
    - This logics grounding allows the user to interact with the model more naturally, as opposed to having to prompt or define every concept from scratch.
    - Furthermore, ChatGPT’s out-of-the-box understanding of basic robotics concepts such as control, camera geometry, and physical form factors makes it an excellent choice to build upon for generalizable and user-friendly robotics pipelines.

Experimental Results (4)

Computer vision problem



Imagine we are working with an RGBD camera. The pose of this camera can be accessed via a function get_pose(). Now let us say we observe a point P in the image at coordinates (x, y), and the corresponding perspective depth value can be obtained by get_depth(x, y). The intrinsics of the camera can be obtained by the function get_intrinsics(). Can you give me code that will compute the world coordinates corresponding to point P?

Sure, here's some example code in Python to compute the world coordinates corresponding to point P

```python
import numpy as np
def get_world_coordinates(x, y):
    # Get camera intrinsics
    K = get_intrinsics()
    # Get camera pose
    pose = get_pose()
    # Get perspective depth value at observed point
    depth = get_depth(x, y)
    # Compute homogeneous image coordinates
    p_image = np.array([x, y, 1])
    # Compute inverse camera intrinsics
    K_inv = np.linalg.inv(K)
    # Compute image coordinates in camera frame
    p_camera = depth * K_inv.dot(p_image)
    # Compute world coordinates
    p_world = np.dot(pose[:3, :3], p_camera) + pose[:3, 3]
    return p_world
```

Given an observed point P in the image at coordinates (x, y), and the corresponding perspective depth value obtained by get_depth(x, y), we can compute the 3D world coordinates corresponding to P. To do this, we need to know the camera intrinsics, which describe how the 3D scene is projected onto the 2D image plane. The intrinsics can be obtained using the get_intrinsics() function.

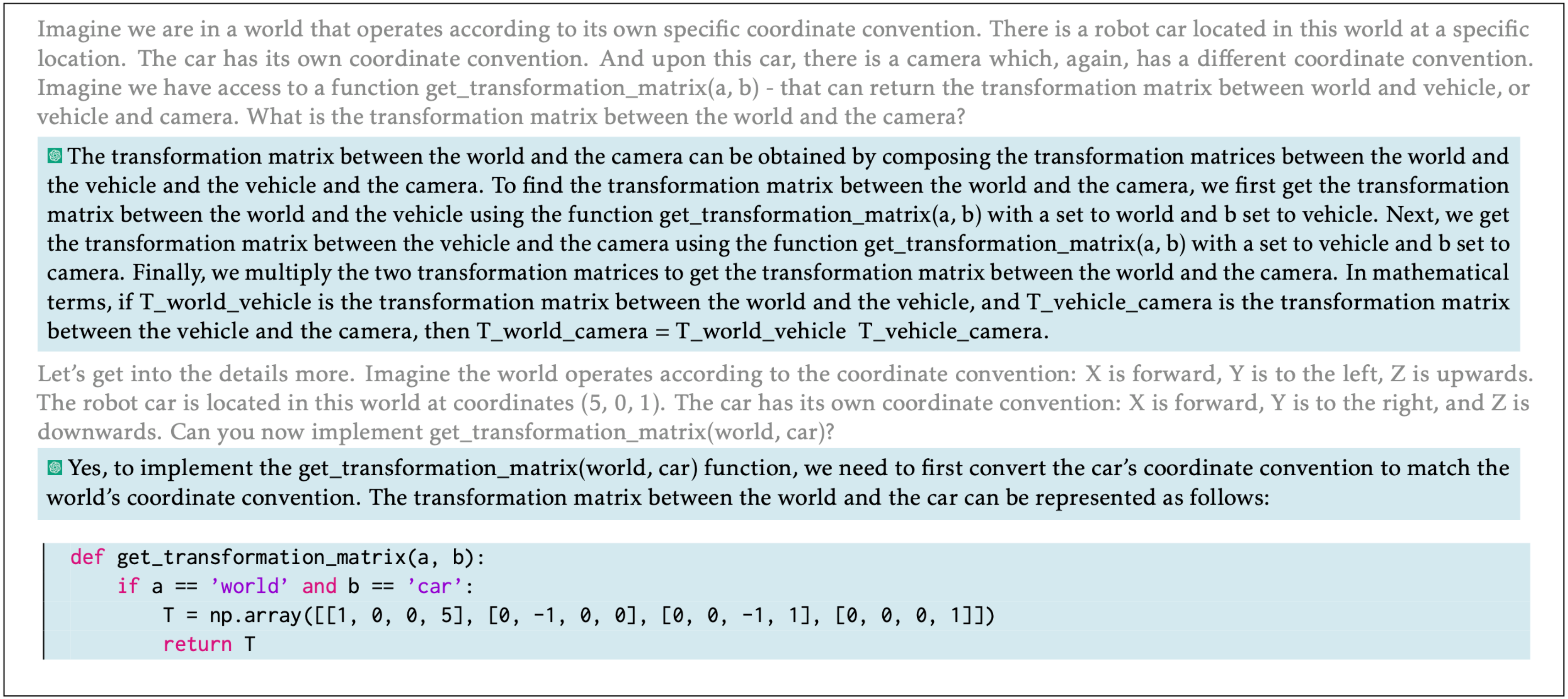
We also need to know the camera pose, which describes the position and orientation of the camera in the world coordinate system. The pose can be obtained using the get_pose() function.

With the intrinsics and pose, we can transform the observed point P from image coordinates to world coordinates. We first compute the inverse of the camera intrinsics matrix, which allows us to convert from image coordinates to camera coordinates. We then multiply the inverse intrinsics matrix with the observed point's pixel coordinates and the corresponding depth value to obtain the 3D point in camera coordinates.

Finally, we transform the camera coordinates to world coordinates by multiplying the camera coordinates by the camera pose matrix, which describes the transformation from camera coordinates to world coordinates. The resulting 3D point is in homogeneous coordinates, so we need to divide by the last coordinate to obtain the 3D Euclidean coordinates.

Experimental Results (4)

Transformation matrices



Imagine we are in a world that operates according to its own specific coordinate convention. There is a robot car located in this world at a specific location. The car has its own coordinate convention. And upon this car, there is a camera which, again, has a different coordinate convention. Imagine we have access to a function get_transformation_matrix(a, b) - that can return the transformation matrix between world and vehicle, or vehicle and camera. What is the transformation matrix between the world and the camera?

The transformation matrix between the world and the camera can be obtained by composing the transformation matrices between the world and the vehicle and the vehicle and the camera. To find the transformation matrix between the world and the camera, we first get the transformation matrix between the world and the vehicle using the function get_transformation_matrix(a, b) with a set to world and b set to vehicle. Next, we get the transformation matrix between the vehicle and the camera using the function get_transformation_matrix(a, b) with a set to vehicle and b set to camera. Finally, we multiply the two transformation matrices to get the transformation matrix between the world and the camera. In mathematical terms, if T_world_vehicle is the transformation matrix between the world and the vehicle, and T_vehicle_camera is the transformation matrix between the vehicle and the camera, then T_world_camera = T_world_vehicle T_vehicle_camera.

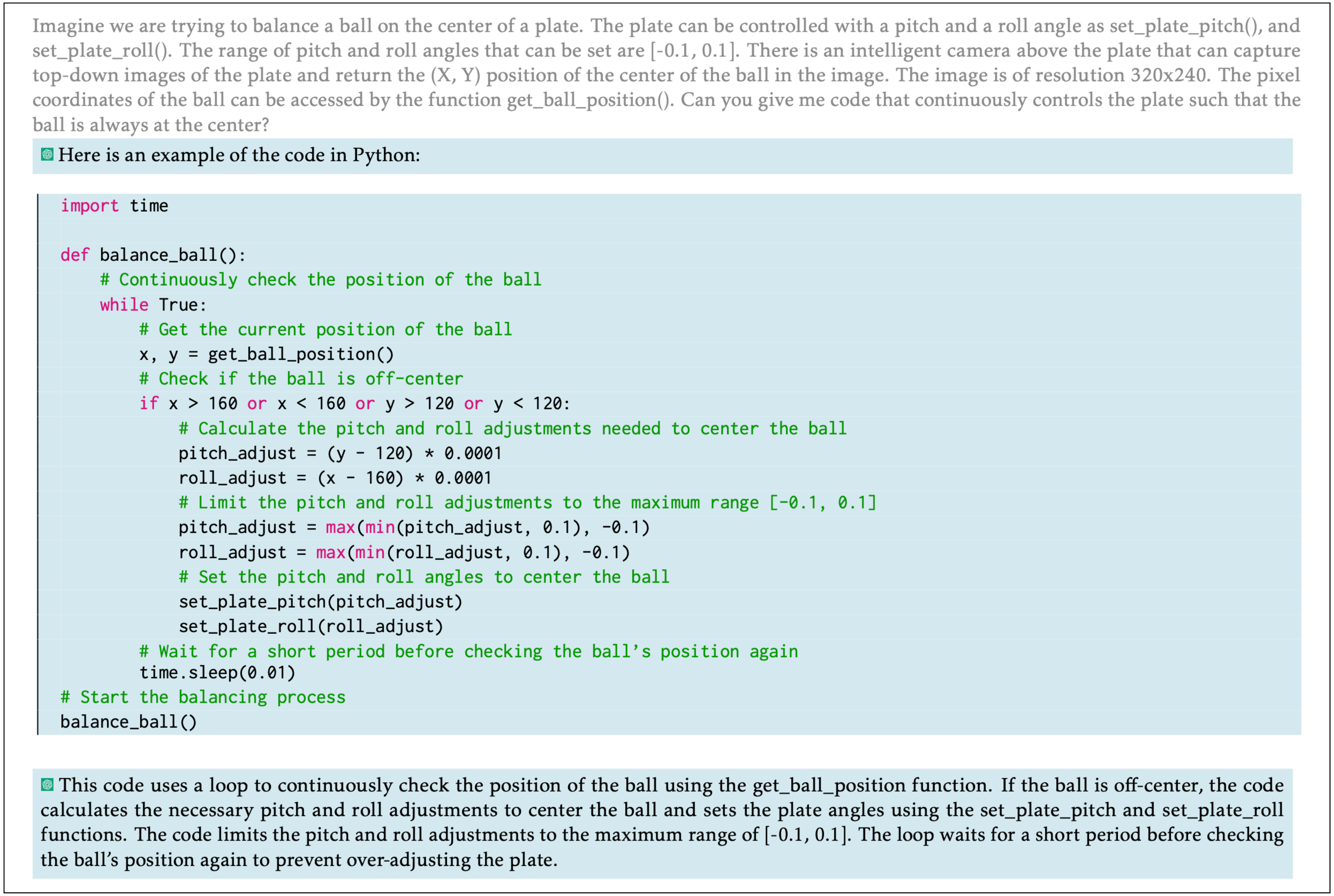
Let’s get into the details more. Imagine the world operates according to the coordinate convention: X is forward, Y is to the left, Z is upwards. The robot car is located in this world at coordinates (5, 0, 1). The car has its own coordinate convention: X is forward, Y is to the right, and Z is downwards. Can you now implement get_transformation_matrix(world, car)?

Yes, to implement the get_transformation_matrix(world, car) function, we need to first convert the car’s coordinate convention to match the world’s coordinate convention. The transformation matrix between the world and the car can be represented as follows:

```
def get_transformation_matrix(a, b):
    if a == ’world’ and b == ’car’:
        T = np.array([[1, 0, 0, 5], [0, -1, 0, 0], [0, 0, -1, 1], [0, 0, 0, 1]])
        return T
```

Experimental Results (4)

Balancing a ball on a plate



Imagine we are trying to balance a ball on the center of a plate. The plate can be controlled with a pitch and a roll angle as set_plate_pitch(), and set_plate_roll(). The range of pitch and roll angles that can be set are [-0.1, 0.1]. There is an intelligent camera above the plate that can capture top-down images of the plate and return the (X, Y) position of the center of the ball in the image. The image is of resolution 320x240. The pixel coordinates of the ball can be accessed by the function get_ball_position(). Can you give me code that continuously controls the plate such that the ball is always at the center?

Here is an example of the code in Python:

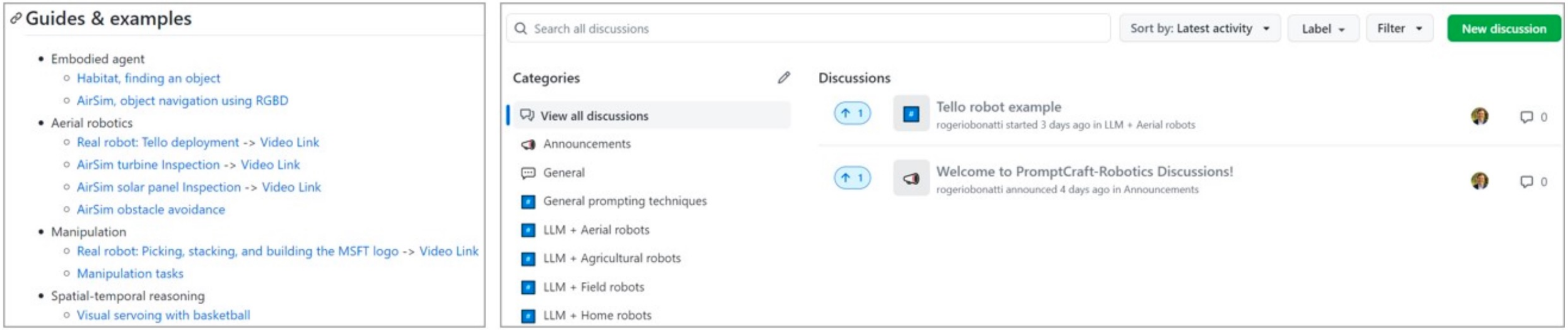
```python
import time

def balance_ball():
    # Continuously check the position of the ball
    while True:
        # Get the current position of the ball
        x, y = get_ball_position()
        # Check if the ball is off-center
        if x > 160 or x < 160 or y > 120 or y < 120:
            # Calculate the pitch and roll adjustments needed to center the ball
            pitch_adjust = (y - 120) * 0.0001
            roll_adjust = (x - 160) * 0.0001
            # Limit the pitch and roll adjustments to the maximum range [-0.1, 0.1]
            pitch_adjust = max(min(pitch_adjust, 0.1), -0.1)
            roll_adjust = max(min(roll_adjust, 0.1), -0.1)
            # Set the pitch and roll angles to center the ball
            set_plate_pitch(pitch_adjust)
            set_plate_roll(roll_adjust)
        # Wait for a short period before checking the ball’s position again
        time.sleep(0.01)
# Start the balancing process
balance_ball()
```

This code uses a loop to continuously check the position of the ball using the get_ball_position function. If the ball is off-center, the code calculates the necessary pitch and roll adjustments to center the ball and sets the plate angles using the set_plate_pitch and set_plate_roll functions. The code limits the pitch and roll adjustments to the maximum range of [-0.1, 0.1]. The loop waits for a short period before checking the ball’s position again to prevent over-adjusting the plate.

PromptCraft

- **A collaborative tool** for LLM + Robotics research
  - Prompt engineering is **particularly challenging at the intersection of LLMs with robotics**, <u>where there is a lack of comprehensive and accessible resources that provide examples of positive (and negative) interactions.</u>
  - For researchers to share examples of prompting strategies and test their algorithms in sample robotic environments



Guides & examples

- Embodied agent
  - Habitat, finding an object
  - AirSim, object navigation using RGBD
- Aerial robotics
  - Real robot: Tello deployment -> Video Link
  - AirSim turbine Inspection -> Video Link
  - AirSim solar panel Inspection -> Video Link
  - AirSim obstacle avoidance
- Manipulation
  - Real robot: Picking, stacking, and building the MSFT logo -> Video Link
  - Manipulation tasks
- Spatial-temporal reasoning
  - Visual servoing with basketball

Discussions

1 Tello robot example
rogeriobonatti started 3 days ago in LLM + Aerial robots 0

1 Welcome to PromptCraft-Robotics Discussions!
rogeriobonatti announced 4 days ago in Announcements 0

Figure 6: Promptcraft open-sourced repository. Researchers can upload and vote on examples of LLM prompts for various robotics categories.

PromptCraft

- a **Github-based platform** that
  - allows researchers to share examples of prompt engineering strategies within different robotics categories, such as navigation, grasping, and manipulation.
  - Users can submit their examples and rate others' submissions, which we hope will create a community-driven resource for researchers working with LLMs.
  - Submissions of prompts and dialogues are primarely based on text, but we encourage users to share videos and images depicting the robot's behavior, especially for real-world deployment scenarios.

- **Offers an AirSim environment with a ChatGPT wrapper** for researchers to prototype prompts and algorithms in a controlled simulated setting.
  - We welcome contributions of new test environments to expand the range of scenarios where researchers can test their algorithms.

- **AI for Science**
  - With Promptcraft we aim to support the empirical science of prompt engineering and enable researchers to advance the field.

Summary

- **Demonstrate a pipeline for applying ChatGPT to robotics tasks.**
  - The pipeline involves several prompting techniques such as free-form natural language dialogue, code prompting, XML tags, and closed-loop reasoning. We also show how users can leverage a high-level function library that allows the model to quickly parse human intent and generate code for solving the problem

- **Experimentally evaluate ChatGPT's ability to execute a variety of robotics tasks.**
  - We show the model's capabilities and limitations when solving mathematical, logical, and geometrical operations, and then explore more complex scenarios involving embodied agents, aerial navigation, and manipulation. We include both simulation and real-world experiments that result from ChatGPT's plans

- **Introduce a collaborative open-source platform, PromptCraft**
  - where researchers can work together to provide examples of positive (and negative) prompting strategies when working with LLMs in the robotics context. Prompt engineering is a mostly empirical science, and we want to provide a simple interface for researchers to contribute with knowledge as a community. Over time we aim to provide different environments where users can test their prompts, and welcome new contributions

- **Release a ChatGPT-integrated simulation tool based on Microsoft AirSim.**
  - This AirSim-ChatGPT simulation contains a sample environment for drone navigation and aims to be a starting point for researchers to explore how ChatGPT can enable robotics scenarios.

Limitations

- These tools should not be given full control of the robotics pipeline, especially for **safety-critical** applications.

- Most of the examples presented demonstrated **open perception-action loops** where ChatGPT generated code to solve a task, with **no feedback** was provided to the model afterwards

ChatGPT for paper writing

- **Please note that this paper was largely written with the assistance of ChatGPT**, with prompts provided by the authors.

- **The model's output was thoroughly revised and adapted**, we note that the use of LLMs can significantly speed up the writing process, and we recommend their use to the interested reader.

Extended Reading

Just say the magic word: using language to program robots

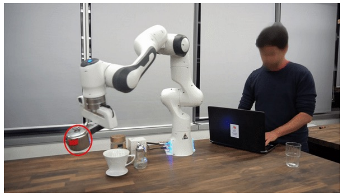
- LaTTe is a deep machine learning model that lets us send language commands to robots in an intuitive way with ease. When given an input sentence by the user, the model fuses it with camera images of objects that the robot observes in its surroundings, and outputs the desired robot behavior.
  - LaTTe [in submission] and Trajectory Language Transformer [IROS 2022]
  - Check out the project's webpage | View our machine learning models on our Github repository

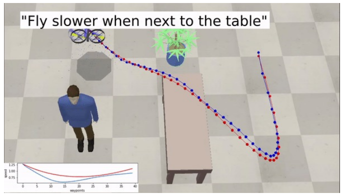
LaTTe: Language Trajectory TransformEr

Submission to CORL 2022

Arthur Bucker[1] , Luis Figueredo[1], Sami Haddadin[1], Ashish Kapoor[2] ,
Shuang Ma[2] , Sai Vemprala[2], Rogerio Bonatti[2]

[1]Technische Universität München (TUM), [2]Microsoft





"Fly slower when next to the table"

Thank you~

Southern University of Science and Technology