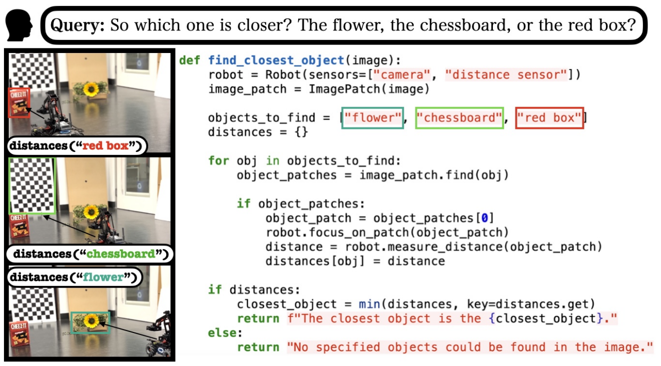
Actions as Programs in the Modern Era

Angelos Mavrogiannis, Yiannis Aloimonos

Abstract— **Perception and planning in robotics have often been treated as separate problems in the literature. We argue that this gap occurs because of the absence of a persistent common representational space for perception and action. We believe that the most efficient representation that encapsulates perception and action is a computer program. A program can incorporate both motoric functions required for planning and control, and perceptual functions that are essential for sensing the environment. When called withing the context of a program, these functions can be combined with expressive programming constructs such as control flow tools, making them suitable for downstream tasks that require decision making, such as planning. We highlight the key advantages of this representation and discuss new research directions that arise from thinking of a robot action as a program in a practical robot planning point of view and in a more algorithmic computer science perspective.**

Perception and Action: The problems of planning and perception were traditionally studied separately by different communities. The modeling of the planning problem occurred on a symbolic level where many mathematical frameworks such as behavior trees, parse trees, graphs, and other mechanisms have been employed to represent the plan of an action [10]. Integrating visual feedback from the environment into a planning problem is challenging, and in practice often involves the underlying assumption that perception is represented as a black box that continuously delivers 3D descriptions of the world with a number of labels attached to them. However, this distinction between perception and planning as two separate processes has not been helpful for the evolution of the field of robotics. As the plan of the action is unraveling, a number of perceptual tests is required at various stages of the computation. Some of these tests ensure that action pre-conditions are satisfied, recognize the current stage of the action so that appropriate motions are generated by the control system, and track the progress of the robot towards achieving a goal. This highlights the fact that perception and action are intertwined, and that one cannot exist without the other in robotics. Based on these observations and inspired by the Theory of Event Coding (TEC) [3], we believe that the functional architecture supporting perception and action planning should be formed by a common representational domain for perceived events (perception) and intended or to-be generated events (action). Motivated by this idea, our insight is that linking actions and percepts by projecting them to a common programmatic representation space leverages the expressiveness of programming constructs and the commonsense reasoning capabilites of Large Language Models (LLMs), and enables us to reason

The authors are with the Department of Computer Science at the University of Maryland, College Park, 8125 Paint Branch Dr, College Park, MD 20742, USA. {angelosm, jyaloimo}@umd.edu


Query: So which one is closer? The flower, the chessboard, or the red box?
distances("red box")
distances("chessboard")
distances("flower")

```python
def find_closest_object(image):
    robot = Robot(sensors=["camera", "distance sensor"])
    image_patch = ImagePatch(image)

    objects_to_find = ["flower", "chessboard", "red box"]
    distances = {}

    for obj in objects_to_find:
        object_patches = image_patch.find(obj)

        if object_patches:
            object_patch = object_patches[0]
            robot.focus_on_patch(object_patch)
            distance = robot.measure_distance(object_patch)
            distances[obj] = distance

    if distances:
        closest_object = min(distances, key=distances.get)
        return f"The closest object is the {closest_object}."
    else:
        return "No specified objects could be found in the image."
```

Fig. 1: A preliminary demonstration of our end-to-end framework actively solving a distance query. The LLM-generated code invokes the `measure_distance` function and the `find` function from the perception-action API. These are wrapper functions that map to lower-level API calls accessing an on-board distance sensor and a camera.

towards executing complex tasks and generate perceptually-grounded action plans. Equipped with a set of logical reasoning constructs such as loops and conditionals, a program serves as an ideal representational medium [7], enabling function calls to Vision-Language Models (VLMs) [6, 11] to segment and classify objects of interest [4], and saving their output to intermediate variables that are consumable downstream. These intermediate variables illustrate a step-by-step explainable reasoning process towards the task at hand, often expressed in simple python pseudocode-resembling grammar [13]. The produced plans are interpretable and flexible in terms of incorporating any vision or language model API as a function call. At the same time, the generated programs benefit from the compositional power of LLMs to decompose high-level tasks into sub-tasks on a lower level of abstraction without the need of any additional training. In Fig. 1 we demonstrate preliminary results of our end-to-end framework that generates action-perception programs for solving simple natural language queries and executes them on a ground navigation robot. The demonstrated program is produced by an LLM prompted with a natural language query (question) and an action-perception API based on ViperGPT [13]. This API consists of motoric and perceptual wrapper functions that map to lower-level controllers and VLM implementations, respectively. The purpose of this program is answering a query about the relative distance of a set of objects to the robot, which is achieved by the call to a set of action and perceptual wrapper functions combined with loops and conditionals in python syntax.

Commonsense Reasoning: The idea of modeling actions as programs has been explored in prior literature [5]. However, composing these programs previously required skilled programmers with domain knowledge, and the produced software was often inadvertently tied to a specific robot or application. The advent of LLMs and their ability to synthesize programs have revolutionized this idea. Leveraging a notion of artificial commonsense reasoning acquired from pre-training on vast internet corpora, these models have enabled new capabilities especially in the domain of robot task planning. Applications that previously relied on existing or manually-crafted knowledge bases to reason about task execution can now obtain such information by querying a language model on demand in real time. Experimental results have shown [12] that the power of LLMs can be leveraged more efficiently in robot task planning when actions and tasks are modeled as programs, compared to natural language representations. This occurs due to the vast presence of code that is publicly available on the web. Considering a causal language modeling objective where an LLM predicts the next word autoregressively, it is harder to reason in the practically infinite space of free-form natural language. On the other hand, programming logic can be found in abundance on the internet and is bound by stricter grammatical and syntactic rules. This makes programs a convenient medium for modeling problems and solving them through modular decision making and counterfactual reasoning through control flow tools and other expressive programming constructs.

Expressiveness: Control flow tools are the structural components of a programming language. Modeling actions as programs inherently facilitates counterfactual reasoning through the imaginary "*if...then...else*" statement. This statement enables decision making and planning that can be triggered by potential events, subsuming the functionality of previously used representations such as binary decision trees. Furthermore, a lot of tasks in robotics are inherently repetitive on a certain level of granularity. For example, a cooking robot instructed to stir a pot might perform the repetitive motion of rotating a spatula multiple times, which can be modeled with the construct of a loop. The action repetition can occur dynamically through a `while` loop with an appropriate stopping condition, or statically for a fixed number of times with a `for` loop. Additional programming concepts such as polymorphism are also applicable to robot planning applications and are currently underexplored. Overloading a function can map to different program implementations for the same task depending on the context and the robot perception of the environment. For example, imagine that we have access to an LLM planner that receives a function name describing a high-level task along with a set of appropriate parameters, and returns a program that implements this task using a set of actions that are admissible in the environment. Then, the program for implementing the function `cook(obj: pasta)` might map to a different implementation than the program for the function `cook(obj: pasta, loc: oven)` as we have demonstrated in prior work [8], where `obj` and `loc` correspond to the direct object and location of the action, respectively. The former program will most likely assume that a pot is available and will generate a sequence of instructions that boil the pasta using a pot, following the most likely pattern seen in the data during the pre-training of the LLM. However, in the case of adding the oven as an extra parameter, the latter program will require cooking the pasta in the oven and will hence map to a different implementation.

Distributability: There is a well-established and widely used functional infrastructure for software version control [1], enabling programmers to share, interactively modify, test, and deploy programs to build software applications. While there have been attempts on gathering crowdsourced programs for planning in robotics, they are focusing on a high-level prompt engineering layer of abstraction [9] which requires additional intermediate modules for mapping high-level plans to lower-level realizable robot controllers. Therefore, there are still exciting opportunities for future research on leveraging this existing infrastructure and adapting it to the programmatic action representation in downstream layers, mapping high-level programs to low-level robot control programs. We believe that this could be a promising direction towards building a large dynamic open-source repository of action programs that are hardware-agnostic and applicable among different platforms and applications.

Theoretical Benefits: One of the key ideas of the minimalist program in linguistics [2] is the optimality of the human language ability with its underlying components reducing to a very simple computation. From an evolutionary perspective, we view the development of action through the prism of language acquisition. Under this assumption, having access to a program that describes an action one can ask further questions about quantifying the optimality of the action as it exhibits itself through the complexity of the program, using a computational complexity metric such as the Kolmogorov complexity. Clearly, when thinking about the cognitive system of an agent, it is often required that this agent behaves in an optimal way and hence that the actions performed by the agent are orchestrated in such a way that the minimum amount of effort or energy is spent. The amount of energy spent by a program can theoretically be computed under the assumption that the cost of all operations is known. We can extend these thoughts about a single action to relationships between multiple actions. For example, a robot programmer can compose three different programs that all implement the same task with different sequences of actions. But which one is the best? What is "*best*" and what are the criteria? Some criteria could be the minimization or maximization of the contact of the robot with the world, or various other constraints and desiderata posed by the problem at hand. It is one thing to say that you can program the robot to perform an action and another for the robot to learn how to reason about which action is better, faster, slower or whatever the attribute is for the comparison. We believe that these ideas, driven by thinking of actions as programs, can give rise to new research directions in the field of robotics, both in theory, and in practice.

References

[1] GitHub. URL https://github.com.

[2] N. Chomsky. *The minimalist program*. MIT press, 2014.

[3] B. Hommel, J. Müsseler, G. Aschersleben, and W. Prinz. The theory of event coding (tec): A framework for perception and action planning. *Behavioral and brain sciences*, 24(5):849–878, 2001.

[4] S. Huang, Z. Jiang, H. Dong, Y. Qiao, P. Gao, and H. Li. Instruct2act: Mapping multi-modality instructions to robotic actions with large language model. *arXiv preprint arXiv:2305.11176*, 2023.

[5] G. Kazhoyan and M. Beetz. Programming robotic agents with action descriptions. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 103–108, 2017. doi: 10.1109/IROS.2017.8202144.

[6] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[7] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[8] A. Mavrogiannis, C. Mavrogiannis, and Y. Aloimonos. Cook2ltl: Translating cooking recipes to ltl formulae using large language models, 2023.

[9] Microsoft. PromptCraft-Robotics, 2023. URL https://github.com/microsoft/PromptCraft-Robotics.

[10] B. Patle, A. Pandey, D. Parhi, A. Jagadeesh, et al. A review: On path planning strategies for navigation of mobile robot. *Defence Technology*, 15(4):582–606, 2019.

[11] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[12] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, 2023.

[13] D. Surís, S. Menon, and C. Vondrick. Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*, 2023.