Meta-Optimization and Program Search using Language Models for Task and Motion Planning

Denis Shcherba[*]
TU Berlin
d.shcherba@campus.tu-berlin.de

Eckart Cobo-Briesewitz[*]
TU Berlin
cobo-briesewitz@campus.tu-berlin.de

Cornelius V. Braun
TU Berlin

Marc Toussaint
TU Berlin

Abstract: Intelligent interaction with the real world requires robotic agents to jointly reason over high-level plans and low-level controls. Task and motion planning (TAMP) addresses this by combining symbolic planning and continuous trajectory generation. Recently, foundation model approaches to TAMP have presented impressive results, including fast planning times and the execution of natural language instructions. Yet, the optimal interface between high-level planning and low-level motion generation remains an open question: prior approaches are limited by either too much abstraction (e.g., chaining simplified skill primitives) or a lack thereof (e.g., direct joint angle prediction). Our method introduces a novel technique employing a form of meta-optimization to address these issues by: (i) using program search over trajectory optimization problems as an interface between a foundation model and robot control, and (ii) leveraging a zero-order method to optimize numerical parameters in the foundation model output. Results on challenging object manipulation and drawing tasks confirm that our proposed method improves over prior TAMP approaches.

Keywords: Task and Motion Planning, LLMs as Optimizers, Trajectory Optimization

1 Introduction

Intelligent interaction with the world requires flexible action plans that are robust and satisfy real-world constraints. To achieve a long-horizon goal, agents are required not only to reason over symbolic decisions, but also to make geometry-based decisions to seamlessly integrate high-level reasoning and low-level control. The resulting decision-making problems are challenging to solve due to the combinatorial complexity of the search space with increasing plan lengths.

Commonly, the challenges of these decision-making problems are approached through the lens of integrated task and motion planning (TAMP) [1, 2, 3, 4]. The core idea of TAMP is to make long-horizon sequential manipulation planning tractable by introducing a symbolic domain that links discrete high-level decisions with continuous motions [5]. Over the past decades, this framework has been shown capable of solving a wide variety of tasks – from table-top manipulation [6, 7, 8], to puzzle solving [9], to architectural construction [10].

While highly general, classical TAMP approaches trade expressivity for tractability: they fix the mapping between symbolic skills and the underlying trajectory optimization, a function that must be designed by an expert engineer. This imposes two key limitations: First, solution quality depends on the designer's insight and experience. Second, the flexibility of these methods is severely limited

[*]Equal contribution Project page: https://mops-tamp.github.io/

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

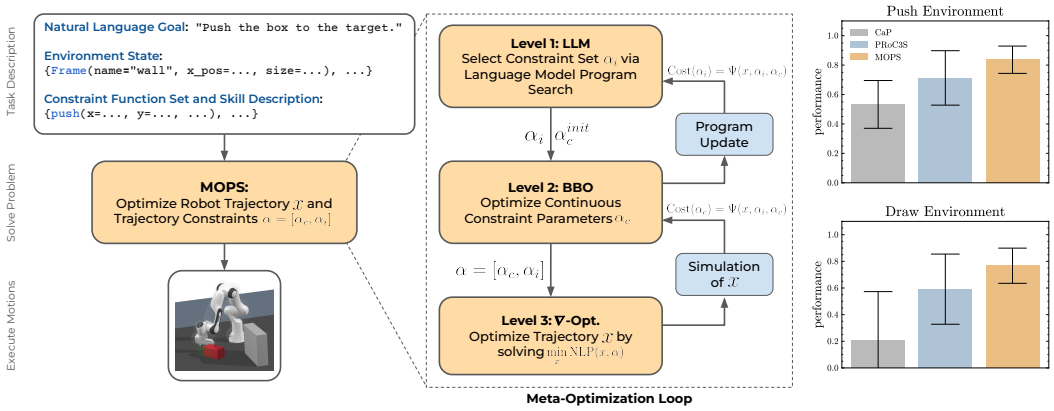


Figure 1: Overview diagram of our method MOPS and its empirical performance. **Left:** MOPS solves the problem by iterating a meta-optimization loop: an LLM optimizes the selection of constraints, a blackbox optimizer (BBO) improves the continuous parameters of these functions and a gradient-based method solves the induced non-linear program (NLP) for a trajectory, which is then simulated to compute the full cost. Based on the back-propagated costs, the constraints of the NLP are adapted until convergence. **Right:** Average normalized performance across two domains, with three tasks each. MOPS outperforms prior methods that search over action sequences (PRoC3S) or simple code snippets (CaP). Results for each task are averaged over 5 independent runs (± 1.96 standard deviation).

because the reasoning over NLPs occurs only at the symbolical level, while the specific timings, scalings, and other parameters of the trajectory constraints are abstracted away during planning. This rigid separation often leads to plans that are only locally optimal or not feasible at all [1, 11].

Recently, foundation model (FM)-based methods have been presented as an alternative to classical TAMP methods, since they do not require hand-crafted symbolic predicates to solve tasks. Instead, they can select control policies from a pre-trained repertoire [12], write code that controls the robot [13], or directly predict joint states [14] given a high-level goal specification.

While FMs have now been applied successfully to a wide range of tasks, most prior approaches are built on simplifying and restrictive assumptions. A common limiting factor is their reliance on atomic skills, originating from the lack of fine-grained spatial reasoning in current foundation models [13, 15, 16, 17]. This allows them to solve simple pick-and-place tasks, but fails for tasks that require precise placement locations or low tolerances for errors. Curtis et al. [11] address this shortcoming by formulating language-instructed TAMP as a constraint satisfaction problem. To solve this problem, the authors propose using LLMs to reason over parameterized skills, for which the parameters can be sampled so that the specified constraints hold during manipulation. However, their approach requires human users to specify relevant task constraints and uses uniform sampling to obtain skill parameters. The result is sample-inefficient rejection sampling [18], and an approach that heavily hinges on the capabilities of the user to specify all relevant constraints upfront, which can be challenging for many real-world tasks.

In this paper, we introduce *Meta-Optimization and Program Search using Language Models for TAMP (MOPS)*, a novel approach that overcomes these limitations by treating the TAMP problem as a *meta-optimization* problem in which both the motion constraints and the resulting trajectories are explicit optimization targets. Concretely, we solve this problem by nesting three optimization levels. In the first step, we use an FM as an optimizer [19] to propose parameterized constraint functions $c(x, \alpha_c)$ that define a nonlinear mathematical program (NLP) and sensible initialization heuristics for each numerical constraint parameter in a plan. Second, we perform black-box optimization to optimize the numerical constraint parameters α_c for overall task success and reward. Third, we perform gradient-based trajectory optimization. This step solves the fully defined NLP to produce smooth, and collision-free trajectories that satisfy all task constraints $c(x, \alpha_c)$.

Our contributions are summarized as follows:

- We propose a novel perspective on TAMP by formulating language conditioned TAMP as a search over constraint sequences instead of actions sequences (Section 2.1).
- We introduce a novel multi-level optimization method for sequential robotic manipulation that combines foundation models with parameterized NLPs and gradient-based trajectory optimization, which enables efficient complex robot manipulation (Section 2.3).
- We validate our method, MOPS, on a range of problems and demonstrate that it improves over prior TAMP approaches (Section 4.2).

2 Meta-Optimization and Program Search using Language Models for TAMP

This section introduces our method, MOPS (Meta-Optimization and Program Search), a hierarchical framework that casts language conditioned TAMP as a meta-optimization problem. At the top level, a foundation model (FM) performs a semantic search over discrete constraints; at the middle level, a black-box optimizer refines continuous constraint parameters; and at the lowest level, a gradient-based solver produces a smooth trajectory.

2.1 Problem Formulation

We start by defining the language-conditioned TAMP setting, which follows the general TAMP problem structure [1], but assumes that the task goal is specified in natural language instead of a planning language such as PDDL.

A language-conditioned TAMP problem is specified by the tuple $(\mathcal{S}, \mathcal{C},\ s_0,\ G,\ \mathcal{J},\ g,\ h)$. Here, $\mathcal{S}$ denotes the fully observable state space, which includes robot end effector, and object poses, as well as geometry dimensions and colors (see Fig. 2 for an example). Further, $\mathcal{C} = \mathbb{R}^n \times \mathrm{SE}(3)$ denotes the configuration space of the scene with an n-dof robot, and s_0 denotes the initial state of the task which has natural language goal G.

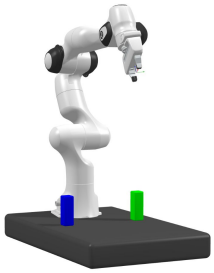
We formulate the language-conditioned TAMP problem following prior work that translates TAMP into constrained optimization problems [2]. Given a natural-language goal G, the objective is to optimize two types of decision variables: *(i)* the continuous trajectory parameters $x \in \mathbb{R}^{T \times n}$, and *(ii)*, the mixed-integer constraint parameters $\alpha \in [0,1]^k \times \mathbb{R}^j$. Specifically, $\alpha = [\alpha_i, \alpha_c]$ consists of binary selectors $\alpha_i \in \{0,1\}^k$ indicating *which* constraints from a predefined library shall be enforced, and continuous parameters $\alpha_c \in \mathbb{R}^j$ specifying their numerical values (e.g., grasp or placement poses). This approach stands in stark contrast to classical TAMP solvers. While those search over a sequence of *actions* (e.g., relating to a PDDL), our approach uses LLMs to search over a set of *constraints* that specify a motion NLP. Following the TAMP and trajectory optimization literature, we denote inequality constraint maps by g and equality constraints by h [2]. For fixed α, the constraint functions are $g(\cdot, \alpha) : \mathbb{R}^{T \times n} \to \mathbb{R}^{m_\alpha}$ and $h(\cdot, \alpha) : \mathbb{R}^{T \times n} \to \mathbb{R}^{p_\alpha}$, where m_α, p_α depend on α. Both g and h can be nonlinear and nonconvex constraints, and we only assume that these maps are (piecewise) continuously differentiable in x.

The full optimization objective is to find a set of plan parameters x, α, such that the overall task cost $\mathcal{J}$ is minimized and that all task constraints are satisfied. Formally, the goal is to optimize

$$\min_{x,\alpha} \mathcal{J}(x, \alpha) = \int_0^T \big[f(x(t), \alpha) + \Psi(x(t), \alpha) \big] \, \mathrm{d}t \tag{1}$$

$$\begin{aligned} \text{s.t.} \quad & g\big(x(t), \alpha\big) \leq 0, \quad \forall t \in [0, T], \\ & h\big(x(t), \alpha\big) = 0, \quad \forall t \in [0, T]. \end{aligned} \tag{2}$$

Following the trajectory optimization literature, we assume that $\mathcal{J}$ is a linear combination of continuous, and differentiable trajectory costs $f(x, \alpha)$, and an *extrinsic* cost $\Psi(x, \alpha)$ which we treat as a black box. In practice, we use $f(x(t), \alpha) = \ddot{x}(t)^T \ddot{x}(t)$, i.e., we minimize squared accelerations.



```
Goal: "Stack the two blocks on top of each other."

State:
{
  Frame(name="block_blue", x_pos=-0.15, y_pos=0., z_pos=0.71, x_rot=0.,
  y_rot=0., z_rot=0., size=[0.04, 0.04, 0.12], color="[255, 0, 0]"),
  Frame(name="block_green", x_pos=0., y_pos=0., z_pos=0.71, x_rot=0.,
  y_rot=0., z_rot=0., size=[0.04, 0.04, 0.12], color="[0, 255, 0]"),
  Frame(name="l_gripper", x_pos=0., y_pos=0.28, z_pos=1.27, x_rot=0.5,
  y_rot=0., z_rot=1.29, size=[0.03], color="[229, 229, 229]"),
  Frame(name="table", x_pos=0., y_pos=0., z_pos=0.6, x_rot=0.,
  y_rot=0., z_rot=0., size=[1.0, 1.0, 0.1], color="[76, 76, 76]")
}
```

(a) Environment State **(b)** User LLM prompt

Figure 2: Illustration of the state definition and user goal description that we prompt the LLM with. (a) Visualizes the environment state, and (b) the corresponding prompt of the LLM, specifying the initial state s_0 as Python dictionary and the planning goal G in natural language.

The extrinsic costs Ψ quantify the degree of task success of a trajectory solution, and depend on the domain (we release our code including all cost functions in the supplementary materials). By decoupling the optimization of f and Ψ, we can exploit specialized solvers for each term.

2.2 The Challenges of Language-Conditioned TAMP

Leveraging FMs for TAMP introduces two key difficulties: First, it is well known that foundation models struggle at geometrical and numerical reasoning [13, 11, 12]. As a result, it is commonly infeasible to synthesize low-level control signals directly from the foundation model. To address this issue, one needs to introduce abstraction layers to the problem. In our case, we take inspiration from recent work by Curtis et al. [11] in separating discrete constraint sampling and continuous parameter sampling. Second, the optimization problem in Eq. (1) is difficult to solve due to its mixed integer structure, black-box extrinsic cost Ψ and high dimensionality. In particular, since many constraints can be selected, each of which depends on multiple parameters, optimizing all variables concurrently is infeasible. For instance, consider the pushing task depicted in Fig. 1. In this task, the robot must not only use the correct set of trajectory constraints to enforce the desired motion, but it also must specify multiple specific poses overtime to fully specify a successful push. We therefore propose to solve the problem by performing a multi-level meta-optimization, which solves each part of the problem leveraging an appropriate method.

2.3 MOPS: Breaking Down the Problem into Three Levels

MOPS (Meta-Optimization and Program Search) frames the language-conditioned TAMP problem of Eq. 1 as a meta-optimization over a Language Model Program [13, LMP], i.e., a parameterized non-linear program (NLP). We denote such a program by NLP(x, α) in the following. Rather than solving discrete constraint selection, continuous parameter optimization, and trajectory synthesis in a single step, MOPS interleaves them in an iterative loop: a foundation model proposes α_i and an initial guess α_c^{init}; a black-box optimizer refines α_c via simulation-based evaluations of the extrinsic cost Ψ; and a gradient-based solver computes the smooth trajectory x under the instantiated constraints. By repeating these complementary stages, MOPS jointly refines both the structure and the parameters of the NLP, driving down the overall cost $\mathcal{J}$. The full approach is illustrated in Fig. 1. We now describe each component in detail.

Level 1: Language Model Program Search. The goal of this stage is to optimize the set of constraints α_i that shall be enforced in the NLP. Thus, the objective is to find the discrete selector α_i that minimizes the extrinsic cost, i.e.,

$$\min_{\alpha_i} \text{Cost}(\alpha_i) = \Psi(x, \alpha_i, \alpha_c), \tag{3}$$

To generate the optimal NLP w.r.t. α_i, we prompt a foundation model with a textual description of the state space $\mathcal{S}$, the initial state s_0, the available constraint functions, and two in-context examples for a different task (we list our prompts in Appendix D). The model is then asked to return: First, a parameterized NLP generation function, which implements constraints selected by the discrete variables $\alpha_i \subset \alpha$. This NLP can be solved for a trajectory x once it is fully parameterized by its continuous parameters α_c. Second, the FM is tasked to return an initial guess α_c^{init} for the continuous parameters α_c, which we further optimize at level two. This prompt structure roughly follows that of Curtis et al. [11], but instead of generating the bounds of a uniform sampling domain, we query the FM for an initial guess for the optimizer.

Level 2: Constraint Parameter Optimization. At this stage, we optimize the continuous constraint parameters of the NLP in Eq. 1, given a set of constraint selection α_i from level 1. To perform this optimization, we iteratively evaluate each set of NLP parameters $\alpha = [\alpha_i, \alpha_c]$ by solving the NLP for the motion x. Subsequently, we execute the solution in simulation to evaluate $\text{Cost}(\alpha_c) = \Psi(x, \bar{\alpha}_i, \alpha_c)$, where $\bar{\alpha}_i$ indicates that α_i is fixed at this level. Since the non-differentiable task-cost Ψ is commonly fast to evaluate, we can use a generic black-box optimizer [20] to optimize the continuous trajectory optimization parameters α_c by iteratively solving the corresponding NLPs at the lowest level.

Level 3: Gradient-Based Trajectory Optimization. The goal of this step is to optimize the robot trajectory x by solving a fully parameterized NLP. Given $\alpha = [\alpha_i, \alpha_c]$, we solve

$$\min_x \int_0^T f(x(t), \alpha)\, \mathrm{d}t \qquad \text{s.t.} \qquad g(x(t), \alpha) \leq 0, h(x(t), \alpha) = 0 \; \forall t \tag{4}$$

using a second-order Augmented Lagrangian method to produce a smooth joint-state trajectory x. We then roll out x inside a physical simulator to obtain the full task cost $\mathcal{J}(x, \alpha)$, which is then used to further refine the higher-level decision variables α.

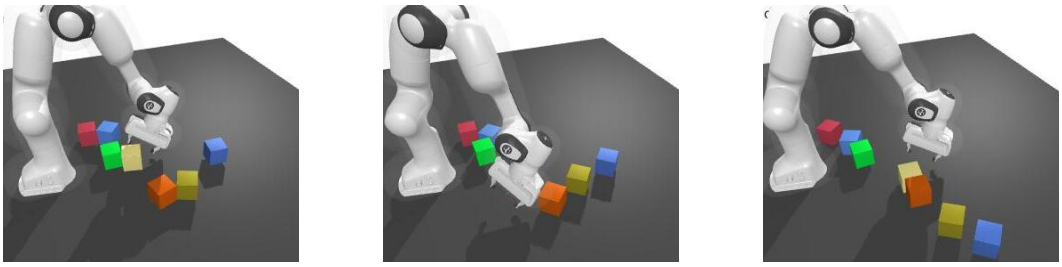
Closing the Loop. We repeat these three levels for a specified number of steps, or until the optimization has converged. Closing the loop enables us to optimize the discrete constraint set selection α_i using the FM. To achieve this, we provide the foundation model with feedback about the outcome of the lower level stages. In practice, we return information about *(i)* the lowest trajectory cost that was found, *(ii)* the final state after running the trajectory, and *(iii)* a target cost that should be achieved for convergence. Closing this loop enables to leverage the FM as a black-box optimizer, as it continuously improves constraint proposals based on prior cost values.

3 Related Work

Task and Motion Planning. The field of Task and Motion Planning (TAMP) [1] has generated various powerful methods for completing complex manipulation tasks in zero-shot manner. What unifies all TAMP approaches is that they solve the task by combining symbolic and discrete reasoning and continuous motion planning [5, 2, 4]. While this approach is highly general, it requires the definition of a fixed set of predicates for reasoning, which can be very challenging in practice [21]. A further challenge in TAMP is the combinatorial complexity of the search space during planning. Therefore, recent work resorted to learning-based approaches [22, 23, 21, 24]. In particular, multiple works replaced traditional search-based planners by LLMs [25, 26, 11]. Our work follows this approach, as we also use a foundation model to optimize the task plan.

Foundation Models in Robotics. LLMs are increasingly applied to robot control and planning [27, 28]. Still, it remains an ongoing field of research to determine an optimal interface between the foundation model and the robot. Some methods sequence learned policies via natural language, effectively using LLMs as a drop-in replacement for a task planner [26, 12, 29, 30]. Other works prompt or fine-tune LLMs to directly predict low-level control signals [14, 31, 32, 33]. However, natural language lacks the required geometric precision for many tasks, and numerical control

```
Goal: "Push the blocks so that they form a straight line."
```



(a) Code as Policies **(b)** PRoC3S **(c)** MOPS (Ours)

Figure 3: Solutions produced by all evaluated methods in the 'Pushing' domain. MOPS is the only method that achieves a straight line that includes all blocks.

outputs are difficult to predict for LLMs. Therefore, code has emerged as a promising paradigm to link foundation model and control policy. Examples include reward function optimization using LLMs [34, 35, 36], writing robot policy code directly [13, 37, 38, 39], and proposing constraints for trajectory optimization which can be used to generate trajectories [40, 11]. Our method follows this last approach, as we also use a foundation model to optimize trajectory constraints. In difference to prior work like PRoC3S [11], however, we optimize the proposed constraints with a zero-order numerical optimization method to further improve the geometric accuracy of the optimization problem.

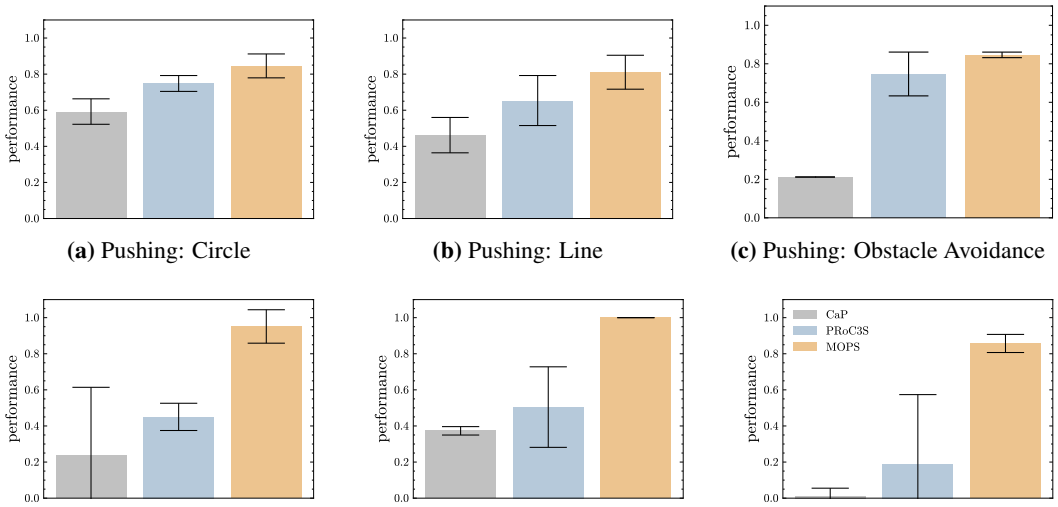
Foundation Models for Optimization. Large pretrained neural networks are commonly referred to as foundation models (FMs) [41]. While most models are still commonly referred to as *language* models, recent work has demonstrated that FMs can serve as versatile optimizers in various contexts. Examples of this include regression or sequence completion [42], code optimization [43, 44], numerical optimization [45], as well as mathematical program optimization [46] and problem solving [47]. At the heart of these FM-based optimization approaches lies iterative prompting and an external fitness or cost function that quantifies the quality of a solution. These ingredients establish a strong connection between evolutionary computation and FMs [19], as both frameworks continuously modify a set of solution candidates to improve their expected performance. Our work follows this framework, but applies within the field of robotics, as we use an LLM to continuously improve a trajectory optimization problem that solves a specified task.

4 Experiments

The goal of this section is to answer three central questions: *(i)* how does MOPS compare to prior language-conditioned TAMP methods?; *(ii)* what role does the black-box optimizer play in our approach?; *(iii)* how important is the LLM feedback loop?

4.1 Experimental Setup

Task Environments. We evaluate our method across two distinct domains. In the *Pushing* environment, a Panda robot must push multiple cuboids into predefined goal configurations. This environment features three specific tasks: *(i)* arranging blocks in a circle, *(ii)* arranging blocks in a straight line, and *(iii)* maneuvering a block around a wall to a target position. The integration of static and dynamic obstacles—such as a wall and other movable blocks—significantly elevates the planning complexity by constraining feasible trajectories, thereby requiring the development of more sophisticated manipulation strategies that account for environmental constraints.



(a) Pushing: Circle **(b)** Pushing: Line **(c)** Pushing: Obstacle Avoidance

(d) Drawing: Star **(e)** Drawing: Pentagon **(f)** Drawing: Hash

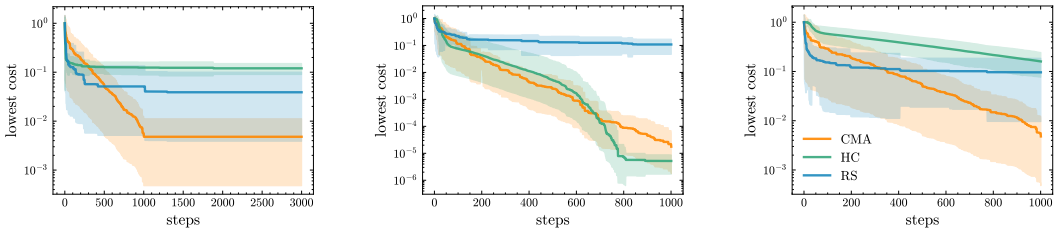
Figure 4: Normalized performances across six challenging tasks. MOPS outperforms the baselines across all tasks. We report averaged normalized performances across 5 independent seeds (± 1.96 standard deviations).

In the *Drawing* environment, a Panda robot must draw on a whiteboard of fixed dimensions but variable tilt angle. We adapted this environment from Curtis et al. [11], but made it considerably more challenging by introducing a tilt to the whiteboard. A top-down camera, aligned with the table frame, captures visual observations of the scene. The objective is to produce symbols that appear visually accurate despite the perspective distortion induced by the angular mismatch between the camera and whiteboard reference frames. We evaluate performance on three specific drawing tasks: (1) a five-pointed star, (2) a regular pentagon, and (3) an the hash character #.

Baseline Methods. We compare against two leading language-conditioned TAMP approaches. As discussed above, a common line of work on language-conditioned TAMP uses the FM to directly generate executable program code. Among these methods, we adopt Code-as-Policies (CaP) [13] as a state-of-the-art representative. CaP leverages FMs to generate complete policy code, including both high-level decision logic and auxiliary helper functions that interface with predefined Python functions for perception, planning, and control. An alternative is to optimize the generated plan in a closed-loop manner, as we do. The closest method in this vein is PRoC3S [11], which closes the plan-generation loop but uses a uniform sampler to produce continuous action parameters. In contrast, MOPS searches over sequences of parameterized *constraints* and tunes their parameters via black-box optimization rather than random sampling. Although we aggregate multiple constraints into compact building blocks to reduce dimensionality in our experiments, MOPS can reason at the level of individual constraints in principle. This stands in stark contrast to approaches that require action abstractions for planning. To evaluate the efficacy of these improvements, we therefore adopt PRoC3S as an additional baseline method.

4.2 Analyzing Performance

To answer how MOPS compares to prior work, we evaluate MOPS and all baselines on all presented tasks. For PRoC3S and our approach, we allocate $1{,}000$ sampling/optimization steps per LLM query in the drawing domain and $1{,}500$ in the block pushing domain due to its more challenging nature. A maximum of 2 feedback iterations were permitted, limiting the total FM prompts to 3. Further experimental details are elaborated in the Appendix. The results of the experiments are depicted in Fig. 4, and an aggregation across domains can be found in Fig. 1. We observe that our method



(a) Pushing: Circle **(b)** Drawing: Pentagon **(c)** Drawing: Star

Figure 5: Results comparing different BBO methods for constraint parameter optimization: CMA-ES, probabilistic hill climbing (HC), and random sampling (RS). We observe that CMA-ES performs the best across tasks. Random sampling can be efficient if the initial guess is good, but fails if this is not the case. We report averaged normalized performances across 5 independent seeds (± 1.96 standard deviations). In addition, we report further results across all tasks are provided in the Appendix 8.

outperforms the baselines on all tasks, or is at least as good as them. In the challenging drawing domain, it is apparent that optimization-free approaches such as CaP cannot solve any tasks, as the initial guess of the FM for the action sequences and precise motions lacks precision. Further, we observe that our method improves over sampling-based prior work (PRoC3S) by introducing an optimization step of the constraint parameters. This is particularly visible in Fig. 3 which displays the qualitative results in the pushing domain. Due to the number of blocks in this domain, the search space dimension for numerical parameters is too high for methods that employ random sampling.

4.3 What matters in MOPS?

To understand the contribution of individual components within our proposed method, we conduct an ablation study that evaluates the continuous parameter optimization. Specifically, we systematically investigated the role of the inner blackbox optimization loop by comparing three distinct optimization strategies: random sampling (as in PRoC3S), CMA-ES [20], and a probabilistic variant of hill climbing that explores parameter space through directional perturbations [48]. Results for three representative tasks are shown in Figure 5. The experiments demonstrate that optimization clearly improves the initial guess from the LLM. While random sampling performs well when the initial guess from the LLM is of high quality, it falls short of achieving substantial improvements on the more challenging tasks. In contrast, CMA-ES rapidly identifies the appropriate constraint parameters to minimize the task cost. Further, we analyze the role of the outer feedback loop in Fig. 6. We see that LLM feedback is crucial for reliable problem solving. While the initial constraint set is sufficient on some runs, the variance across runs is high when no feedback is used and low after 2 rounds of feedback. We expand this analysis in Appendix A.3.

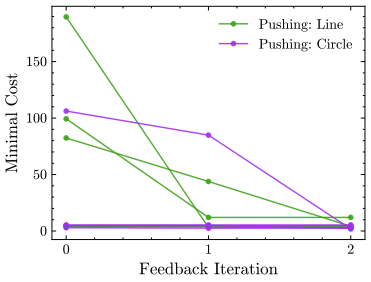


Figure 6: Cost over feedback iterations across 5 independent seeds per task.

5 Conclusion

In this work, we present Meta-Optimization and Program Search (MOPS), a method for TAMP that optimizes sequences of constraints that induce motions to satisfy a language instructed goal. In contrast to prior work, MOPS meta-optimizes trajectory optimization programs using a mix of LLM-, and numerical black-box optimization, enabling it to solve complex manipulation tasks. We conduct a comprehensive evaluation across multiple tasks in diverse environments. Our results show that MOPS improves over prior TAMP approaches.

6 Limitations

While our method offers flexibility and performance across different robotic tasks, it comes with certain limitations. First, it requires the tuning of additional hyperparameters introduced by the inner optimizer (e.g., the step size in hill climbing or the initial sigma in CMA-ES), which may affect robustness and reproducibility. This issue may be mitigated by making part of the hyperparameters tunable by the foundation model, similar to the initial guess prediction that is already part of the method. Second, our approach assumes the availability of a task-specific cost function Ψ, which must be designed to reflect the desired task outcomes. Future work could explore methods for learning or predicting cost functions to improve generality. Further, our current experiments optimize over functions that aggregate multiple constraints due to the limitations of VLMs at the time. We believe that future model releases will permit to reason directly at the individual constraint level. Lastly, our method relies on full state knowledge of the scene; combining with VLMs or incorporating state estimation techniques could help relax this requirement.

Acknowledgments

This research was funded by the Amazon Fulfillment Technolgies and Robotics team.

References

[1] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4(1):265–293, 2021.

[2] M. Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *IJCAI*, pages 1930–1936, 2015.

[3] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pages 1470–1477. IEEE, 2011.

[4] T. Lozano-Pérez and L. P. Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3684–3691. IEEE, 2014.

[5] D. Driess. *Learning for Sequential Manipulation*. PhD thesis, TU Berlin, Germany, 2024.

[6] T. Xue, A. Razmjoo, and S. Calinon. D-lgp: Dynamic logic-geometric program for reactive task and motion planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14888–14894. IEEE, 2024.

[7] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 639–646. IEEE, 2014.

[8] C. V. Braun, J. Ortiz-Haro, M. Toussaint, and O. S. Oguz. Rhh-lgp: Receding horizon and heuristics-based logic-geometric programming for task and motion planning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13761–13768. IEEE, 2022.

[9] S. Levit, J. Ortiz-Haro, and M. Toussaint. Solving sequential manipulation puzzles by finding easier subproblems. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14924–14930. IEEE, 2024.

[10] V. N. Hartmann, O. S. Oguz, D. Driess, M. Toussaint, and A. Menges. Robust task and motion planning for long-horizon architectural construction planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6886–6893. IEEE, 2020.

[11] A. Curtis, N. Kumar, J. Cao, T. Lozano-Pérez, and L. P. Kaelbling. Trust the PRoc3s: Solving long-horizon robotics problems with LLMs and constraint satisfaction. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=r6ZhiVYriY.

[12] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence. PaLM-e: An embodied multimodal language model. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 8469–8488. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/driess23a.html.

[13] 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.

[14] Y.-J. Wang, B. Zhang, J. Chen, and K. Sreenath. Prompt a robot to walk with large language models. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pages 1531–1538. IEEE, 2024.

[15] 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.

[16] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on robot learning*, pages 287–318. PMLR, 2023.

[17] W. Huang, F. Xia, D. Shah, D. Driess, A. Zeng, Y. Lu, P. Florence, I. Mordatch, S. Levine, K. Hausman, et al. Grounded decoding: Guiding text generation with grounded models for embodied agents. *Advances in Neural Information Processing Systems*, 36:59636–59661, 2023.

[18] G. Casella, C. P. Robert, and M. T. Wells. Generalized accept-reject sampling schemes. *Lecture notes-monograph series*, pages 342–347, 2004.

[19] X. Song, Y. Tian, R. T. Lange, C. Lee, Y. Tang, and Y. Chen. Position: leverage foundational models for black-box optimization. In *Proceedings of the 41st International Conference on Machine Learning*, pages 46168–46180, 2024.

[20] N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18, 2003.

[21] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez. Learning symbolic operators for task and motion planning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3182–3189, 2021. doi:10.1109/IROS51168.2021.9635941.

[22] D. Driess, O. Oguz, J.-S. Ha, and M. Toussaint. Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 9563–9569. IEEE, 2020.

[23] X. Fang, C. R. Garrett, C. Eppner, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox. Dimsam: Diffusion models as samplers for task and motion planning under partial observability. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1412–1419. IEEE, 2024.

[24] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Pérez, L. Kaelbling, and J. B. Tenenbaum. Predicate invention for bilevel planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12120–12129, 2023.

[25] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. Kaelbling, and M. Katz. Generalized planning in pddl domains with pretrained large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 20256–20264, 2024.

[26] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[27] J. Wang, E. Shi, H. Hu, C. Ma, Y. Liu, X. Wang, Y. Yao, X. Liu, B. Ge, and S. Zhang. Large language models for robotics: Opportunities, challenges, and perspectives. *Journal of Automation and Intelligence*, 2024.

[28] R. Firoozi, J. Tucker, S. Tian, A. Majumdar, J. Sun, W. Liu, Y. Zhu, S. Song, A. Kapoor, K. Hausman, B. Ichter, D. Driess, J. Wu, C. Lu, and M. Schwager. Foundation models in robotics: Applications, challenges, and the future. *The International Journal of Robotics Research*, 44(5):701–739, 2025. doi:10.1177/02783649241281508.

[29] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*, pages 1769–1782. PMLR, 2023.

[30] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg. Text2motion: From natural language instructions to feasible plans. *Autonomous Robots*, 47(8):1345–1365, 2023.

[31] J. Y. Zhu, C. G. Cano, D. V. Bermudez, and M. Drozdzal. Incoro: In-context learning for robotics control with feedback loops. *arXiv preprint arXiv:2402.05188*, 2024.

[32] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.

[33] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[34] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. Eureka: Human-level reward design via coding large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=IEduRUO55F.

[35] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, et al. Language to rewards for robotic skill synthesis. In *Conference on Robot Learning*, pages 374–404. PMLR, 2023.

[36] J. Song, Z. Zhou, J. Liu, C. Fang, Z. Shu, and L. Ma. Self-refined large language model as automated reward function designer for deep reinforcement learning in robotics. *arXiv preprint arXiv:2309.06687*, 2023.

[37] K. Burns, A. Jain, K. Go, F. Xia, M. Stark, S. Schaal, and K. Hausman. Genchip: Generating robot policy code for high-precision and contact-rich manipulation tasks. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9596–9603. IEEE, 2024.

[38] Y. Mu, J. Chen, Q. Zhang, S. Chen, Q. Yu, C. GE, R. Chen, Z. Liang, M. Hu, C. Tao, et al. Robocodex: multimodal code generation for robotic behavior synthesis. In *Proceedings of the 41st International Conference on Machine Learning*, pages 36434–36454, 2024.

[39] Z. Hu, F. Lucchetti, C. Schlesinger, Y. Saxena, A. Freeman, S. Modak, A. Guha, and J. Biswas. Deploying and evaluating llms to program service mobile robots. *IEEE Robotics and Automation Letters*, 9(3):2853–2860, 2024.

[40] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. In *Conference on Robot Learning*, pages 540–562. PMLR, 2023.

[41] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[42] S. Mirchandani, F. Xia, P. Florence, B. Ichter, D. Driess, M. G. Arenas, K. Rao, D. Sadigh, and A. Zeng. Large language models as general pattern machines, 2023. URL https://arxiv.org/abs/2307.04721.

[43] R. T. Lange, A. Prasad, Q. Sun, M. Faldor, Y. Tang, and D. Ha. The ai cuda engineer: Agentic cuda kernel discovery, optimization and composition. Technical report, Sakana AI, 2025. URL https://pub.sakana.ai/static/paper.pdf.

[44] C. Morris, M. Jurado, and J. Zutty. Llm guided evolution-the automation of models advancing models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 377–384, 2024.

[45] R. Lange, Y. Tian, and Y. Tang. Large language models as evolution strategies. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 579–582, 2024.

[46] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[47] T. H. Trinh, Y. Wu, Q. V. Le, H. He, and T. Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

[48] F. I. Romeo and A. Sangiovanni-Vincentelli. *Probabilistic hill climbing algorithms: Properties and applications*. Electronics Research Laboratory, College of Engineering, University of . . . , 1984.

[49] N. Hansen, Y. Akimoto, and P. Baudis. Cma-es/pycma. *Version r3*, 2, 2022.

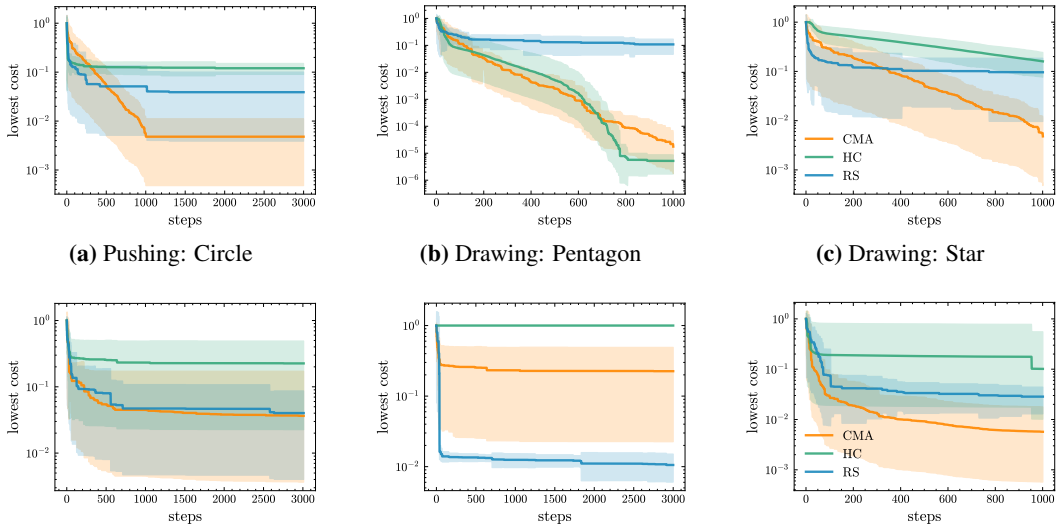
[50] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[51] NVIDIA Corporation. *NVIDIA PhysX SDK, Version 2.8*, 2008. https://developer.nvidia.com/physx-sdk.

A Additional Results

A.1 Optimizer Ablation Study

For completeness, we list the full BBO experiment results across all tasks in Fig. 7. These results complement Fig. 5, which is truncated in the main part owing to space constraints. The full results confirm the results from the main paper. Across four of the six tasks, CMA-ES performs the best. For the pentagon drawing task, hill climbing performs the best. Random sampling performed best for this iteration of the obstacle avoidance task.



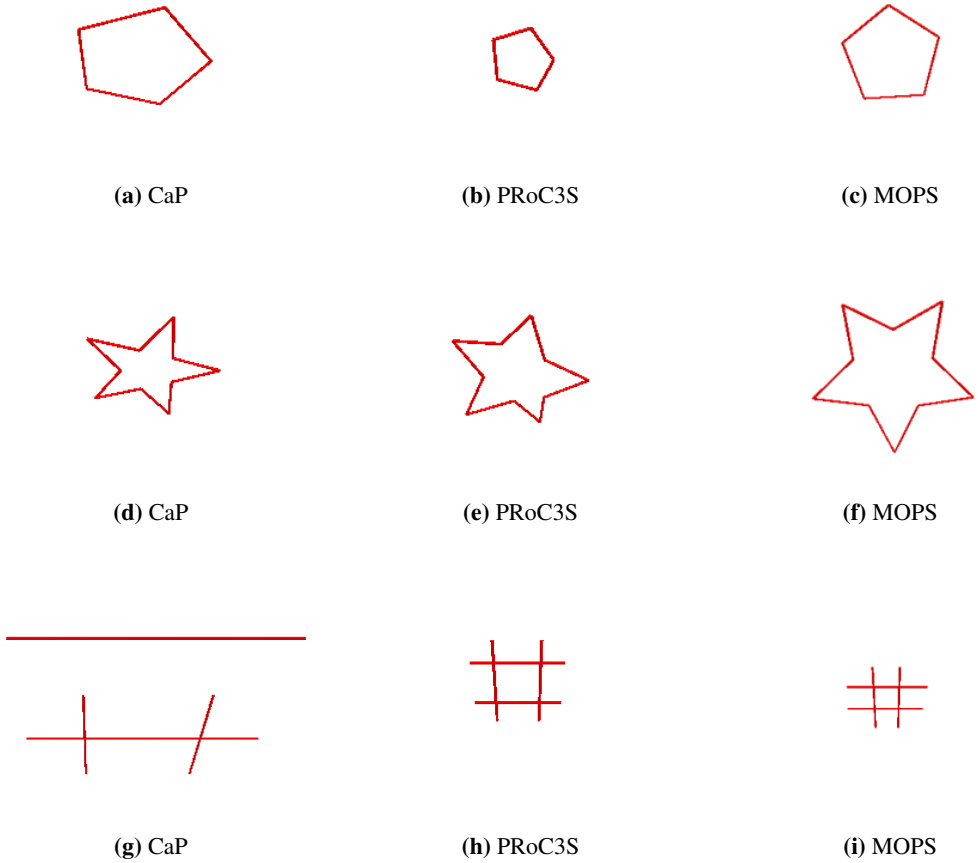
(a) Pushing: Circle **(b)** Drawing: Pentagon **(c)** Drawing: Star

(d) Pushing: Line **(e)** Pushing: Obstacle Avoidance **(f)** Drawing: Hash

Figure 7: Complete ablation study comparing different BBO methods for constraint parameter optimization across all six simulated manipulation tasks. We evaluate CMA-ES, probabilistic hill climbing (HC), and random sampling (RS) on both pushing and drawing tasks. We report averaged normalized performances across 5 independent seeds (± 1.96 standard deviations). The top row shows the main results discussed in Section 4, while the bottom row provides additional task variations that confirm the observed performance trends.

A.2 Qualitative Results for Drawing Tasks

For completeness, we list the qualitative results on the drawing task. That is, we provide the final solutions in Fig. 8. The drawing task evaluates the system's ability to produce visually accurate symbols despite perspective distortion caused by the angular mismatch between the whiteboard tilt angle and the camera's reference frame. The quantitative results in 4 demonstrate that MOPS performs the best on the drawing tasks. The qualitative results illustrate this. The baseline methods, even when provided with complete state information (i.e. camera intrinsics, extrinsics, and global whiteboard position), struggle to produce accurate drawings in the resulting images, highlighting the inherent complexity of this challenge. Our proposed method MOPS significantly outperforms baseline methods by exploiting gradient information within the cost function to optimize line drawing parameters for perceptual accuracy in the image space. By accounting for perspective effects, MOPS generates drawings that look accurate when viewed through the camera.



(a) CaP **(b)** PRoC3S **(c)** MOPS

(d) CaP **(e)** PRoC3S **(f)** MOPS

(g) CaP **(h)** PRoC3S **(i)** MOPS

Figure 8: Resulting images across methods in the drawing environment.

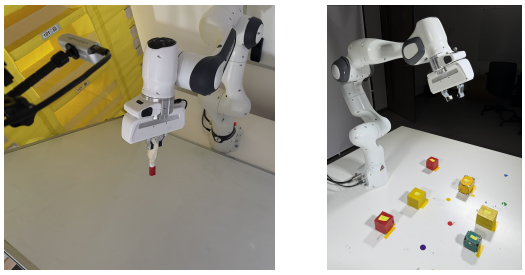
A.3 Detailed Feedback Loop Study

We analyze the contribution of the outer feedback loop, i.e. the language model program search to our method's performance. In the *Drawing* environment, we observed that the outer feedback loop did not significantly improve performance: the FM typically predicted the correct number of lines on the first attempt, after which the inner optimizer converged successfully. Nevertheless, the feedback loop becomes crucial in scenarios where the FM fails to provide a good enough zero-shot guess (e.g., producing an incorrect number of lines).

The importance of the feedback loop becomes clear in the *Pushing* environment. Experimental results, performed across 5 runs per task, highlight its importance for achieving the reported performance (Fig. 6).

A.4 Real-world experiments

To assess the real-world performance of our approach, we deploy it on the Franka Panda robot platform. The experimental setups are shown in Figure 9. While our method transfers successfully to the real robot, careful calibration of the physical setup is required, as simulation provides full state information that is not directly accessible in the real world. Qualitative results and successful executions can be found in the supplementary video.



(a) Drawing **(b)** Block pushing

Figure 9: Real-world experiment setup.

A.5 LLM Ablation Study

We evaluated a wide range of LLMs and VLMs on code generation for our method. For each test, we examined the outputs produced by the models and assessed whether they correctly followed the instructions in the initial prompt. Results were averaged over five runs. The results are shown in Table 1. The most common failure modes were *(i)* hallucinating new high-level functions not specified in the prompt, and *(ii)* omitting key instructions, such as importing required libraries outside the generated functions or creating extra functions when explicitly instructed not to. Performance improvements for smaller models through fine-tuning may be possible; we leave this exploration to future work.

Model	Success Rate
Qwen2-VL-7B-Instruct	0%
Qwen2.5-14B-Instruct-1M	0%
Qwen2.5-Coder-32B-Instruct	0%
Qwen2.5-VL-72B-Instruct	0%
DeepSeek-R1	100%
GPT-3.5-turbo	60%
GPT-4-turbo	100%
GPT-4o-mini	100%

Table 1: Comparison between different LLMs and VLMs.

B Experimental Details

This section lists the full experimental details for this paper.

The code is partially based on Curtis et al. [11]. For the experiments, we use the baseline implementations of CaP and PRoC3S of this repository. For CMA-ES, we use the Python implementation of pycma [49]. All mathematical programs are implemented and optimized via our lab's trajectory optimization library. All experiments we performed on an internal cluster with 12-core CPUs and 32GB of RAM. The code to reproduce our experiments and plots will be made available upon conference publication. For each method we used the OpenAI GPT-4o FM [50], specifically the gpt-4o-mini-2024-07-18 checkpoint. To evaluate the task cost, i.e., success, we simulate the trajectories in the NVIDIA PhysX simulator [51].

Each experiment is repeated 5 times using randomly generated seeds. In each plot, we report the mean performances across all 5 runs and 1.96 standard deviations. To align the cost function scores across tasks, performance is reported as 1 - log-normalized error, calculated by applying $\log(1 + x)$ transformation to raw error values and normalizing by the maximum possible error. Standard deviations undergo identical transformation to preserve scaling consistency. This metric provides a more interpretable performance assessment where higher values indicate better performance, with 1.0 representing perfect execution.

For PRoC3s and our approach, we allocated 1000 sampling/optimization steps per trial in the drawing domain and 1500 in the block pushing domain. A maximum of 2 feedback iterations were permitted, limiting the total FM prompts to 3. For the BBO ablation, we slightly deviate from the general setup and increase the number of independent runs to 10 in the draw environment. To isolate the effect of the optimization method, the initial plan generated by the LLM was fixed across all experimental configurations.

The initial sigma value for the CMA-ES optimizer differs between the two task domains: it is set to 0.01 for the drawing task and 0.05 for the pushing task.

C Cost Function Details

This section provides the cost functions for each of the tasks in both the pushing and drawing domains.

C.1 *Pushing* environment.

Push Box and Avoid Obstacle: Let p_{box} be the position of the box, p_{target} the target position, p_{grip} the gripper position, p_{wall} the wall position, and p_{init} the initial box position. The individual costs are

$$C_{\text{pos}} = \Big(4 \cdot (p_{\text{box}} - p_{\text{target}})\Big)^2 \tag{5}$$

$$C_{\text{wall}} = -\log\Big(\,|\,p_{\text{wall}}^{(xy)} - p_{\text{box}}^{(xy)}\,|\,\Big) \tag{6}$$

$$C_{\text{init}} = -\log\Big(\max\{\,|\,p_{\text{box}} - p_{\text{init}}\,|\,, 0.001\}\Big) \tag{7}$$

$$C_{\text{endeff}} = \Big(4 \cdot (p_{\text{box}} - p_{\text{grip}})\Big)^2 \tag{8}$$

$$C_{\text{endeff-wall}} = -\,|\,p_{\text{wall}} - p_{\text{grip}}\,| \tag{9}$$

The total cost is given by

$$C_{\text{total}} = 2.0\,C_{\text{pos}} + 0.01\,C_{\text{wall}} + 0.01\,C_{\text{init}} + 0.7\,C_{\text{endeff}} + 0.2\,C_{\text{endeff-wall}}. \tag{10}$$

Push Circle: Given points $P = \{p_1, \ldots, p_n\}$ with center $c = \frac{1}{n}\sum_{i=1}^{n} p_i$, we minimize $C_{\text{total}} = 1000 \cdot C_{\text{rad}} + C_{\text{neigh}}$ where $C_{\text{rad}} = \sum_{i=1}^{n}(0.2 - |\,p_i - c\,|\,)^2$ and $C_{\text{neigh}} = \sum_{i=1}^{n}(0.2 - d_i)^2$ with $d_i = \min_{j \neq i} |\,p_j - p_i\,|$.

Push Line: Given points $P = \{p_1, \ldots, p_n\}$ where $p_i = (x_i, y_i)$, we minimize $C_{\text{total}} = 10^4 \cdot C_{\text{fit}} + 10^2 \cdot C_{\text{space}}$ where $C_{\text{fit}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - (\hat{m}x_i + \hat{b}))^2$ for best-fit line $L : y = \hat{m}x + \hat{b}$, and $C_{\text{space}} = \frac{1}{n-1}\sum_{k=1}^{n-1}(d_k - \bar{d})^2$ with $d_k = |\,p'_{(k+1)} - p'_{(k)}\,|$ for consecutive projected points on L and $\bar{d} = \frac{1}{n-1}\sum_{k=1}^{n-1} d_k$.

C.2 *Drawing* environment.

The drawing domain involves more rigorous and fine-tuned cost functions, as it needs to account for connectivity, length, angles, spacings, inner/outer radii, parallelism and further edge cases.

Draw Pentagon: $C_{\text{total}} = C_{\text{conn}} + C_{\text{len}} + C_{\text{rad}} + C_{\text{angle}} + C_{\text{spacing}} + C_{\text{close}} + P_{\text{size}}$ or, together:

$$
\begin{aligned}
C_{\text{total}} = 500 \sum_{i=0}^{4} \left| \, (s_i + v_i) - s_{(i+1) \bmod 5} \, \right|^2 + 500 \frac{\operatorname{Var}(\,|\,v\,|\,)}{\left(\frac{1}{5} \sum_{i=0}^{4} \, |\, v_i \,| \right)^2} \\
+ 300 \frac{\operatorname{Var}(\,|\,s_i - c\,|\,)}{\left(\frac{1}{5} \sum_{i=0}^{4} \, |\, s_i - c \,| \right)^2}, \quad c = \frac{1}{5} \sum_{j=0}^{4} s_j \\
+ 100 \sum_{i=0}^{4} \left[\arccos\left(\frac{-v_{(i-1) \bmod 5} \cdot v_i}{|\, v_{(i-1) \bmod 5} \,| \;\, |\, v_i \,|} \right) - \frac{3\pi}{5} \right]^2 \\
+ 100 \sum_{i=0}^{4} \left[\arccos\left(\frac{(s_i - c) \cdot (s_{(i+1) \bmod 5} - c)}{|\, s_i - c \,| \;\, |\, s_{(i+1) \bmod 5} - c \,|} \right) - \frac{2\pi}{5} \right]^2 \\
+ 200 \, |\, s_0 - (s_4 + v_4) \,|^2 + \mathbf{1} \left[\frac{1}{5} \sum_{i=0}^{4} \, |\, v_i \,| \, < L_{\text{thresh}} \right] \cdot 700
\end{aligned}
$$

Draw Star: $C_{\text{total}} = C_{\text{conn}} + C_{\text{rad,outer}} + C_{\text{rad,inner}} + C_{\text{ratio}} + C_{\text{angle}} + P_{\text{size}}$ or, together:

$$
\begin{aligned}
C_{\text{total}} = 100 \sum_{i=0}^{9} \left| \, (s_i + v_i) - s_{(i+1) \bmod 10} \, \right|^2 \\
+ 500 \frac{\operatorname{Var}\big(\{ \, |\, s_i - c \,| \, : i \in I_{\text{outer}} \} \big)}{(\bar{r}_{\text{outer}})^2} + 500 \frac{\operatorname{Var}\big(\{ \, |\, s_i - c \,| \, : i \in I_{\text{inner}} \} \big)}{(\bar{r}_{\text{inner}})^2} \\
+ \begin{cases} 300(1.5 - \rho)^2, & \rho < 1.5 \\ 100(\rho - 2.0)^2, & \text{otherwise} \end{cases} \quad \rho = \frac{\bar{r}_{\text{outer}}}{\bar{r}_{\text{inner}}} \\
+ 100 \sum_{i=0}^{9} \left[\arccos\left(\frac{(s_i - c) \cdot (s_{(i+1) \bmod 10} - c)}{|\, s_i - c \,| \;\, |\, s_{(i+1) \bmod 10} - c \,|} \right) - \frac{\pi}{5} \right]^2 \\
+ \mathbf{1} \left[\frac{1}{10} \sum_{i=0}^{9} \, |\, v_i \,| \, < 35 \right] \cdot 700
\end{aligned}
$$

where:

$$
\begin{aligned}
c = \frac{1}{10} \sum_{j=0}^{9} s_j, \quad I_{\text{outer}} = \{0, 2, 4, 6, 8\}, \quad I_{\text{inner}} = \{1, 3, 5, 7, 9\}, \\
\bar{r}_{\text{outer}} = \frac{1}{5} \sum_{k \in I_{\text{outer}}} \, |\, s_k - c \,|, \quad \bar{r}_{\text{inner}} = \frac{1}{5} \sum_{k \in I_{\text{inner}}} \, |\, s_k - c \,|
\end{aligned}
$$

Draw hash: $C_{\text{total}} = C_{\text{straight}} + C_{\text{parallel}} + C_{\text{perp}} + C_{\text{spacing}} + C_{\text{intersect}} + C_{\text{len\_con}} + C_{\text{len\_bal}} + P_{\text{size}}$ or, together:

$$
\begin{aligned}
C_{\text{total}} = 100 & \left(\sum_{v \in V_h} \left(\frac{|v_y|}{|\,v\,|} \right)^2 + \sum_{v \in V_v} \left(\frac{|v_x|}{|\,v\,|} \right)^2 \right) \\
& + 50\big((\Delta\alpha_h)^2 + (\Delta\alpha_v)^2\big) \\
& + 50 \sum_{v_h \in V_h} \sum_{v_v \in V_v} \left(\frac{v_h \cdot v_v}{|\,v_h\,|\;|\,v_v\,|} \right)^2 \\
& + 300\big((d_h - 0.33)^2 + (d_v - 0.33)^2\big) + 100(d_h - d_v)^2 \\
& + \sum_{v_h \in V_h} \sum_{v_v \in V_v} \text{Penalty}_{\text{int}}(v_h, v_v) \\
& + 50 \left(\left(\frac{|\,v_{h1}\,| - |\,v_{h2}\,|}{\frac{1}{2}(|\,v_{h1}\,| + |\,v_{h2}\,|)} \right)^2 + \left(\frac{|\,v_{v1}\,| - |\,v_{v2}\,|}{\frac{1}{2}(|\,v_{v1}\,| + |\,v_{v2}\,|)} \right)^2 \right) \\
& + 20 \left(\frac{\sum_{v \in V_h} |\,v\,| - \sum_{v \in V_v} |\,v\,|}{\sum_{v \in V} |\,v\,|} \right)^2 \\
& + \mathbf{1}\big[\bar{l} < 75\big] \cdot 700
\end{aligned}
$$

where:

$V_h = \{v \in V \mid |v_x| > |v_y|\}, \quad V_v = \{v \in V \mid |v_x| \leq |v_y|\}$,
S_h, S_v are the corresponding starting points,

$$d_h = \frac{|s_{h1,y} - s_{h2,y}|}{\bar{l}}, \quad d_v = \frac{|s_{v1,x} - s_{v2,x}|}{\bar{l}}, \quad \bar{l} = \frac{1}{4} \sum_{i=0}^{3} |\,v_i\,|,$$

$\Delta\alpha_h, \Delta\alpha_v$ are the minimum angle differences between parallel vectors,

$\text{Penalty}_{\text{int}}$ is high if segments fail to intersect near $\frac{1}{3}$ or $\frac{2}{3}$ along each segment.

D MOPS Prompting Details

In this section, we present the prompts used in our method, which follow the general structure introduced in PRoC3S [11].

D.1 Prompt for Draw Environment

```
1
2  You are a franka panda robot operating in an environment with the
       following state:
3
4  The whiteboard is bounded in the x-direction between 0 and 0.64, and
       in the y-direction between 0 and 0.48.
5  The whiteboards world position is at (0.0, 0.4, 0.96).
6  The whiteboard is tilted by 40 degrees.
7  A camera, positioned at (0.0, 0.45, 1.5), looks downward parallel to
       the world x-y plane.
8
9
10 You have access to the following set of skills expressed as pddl
       predicates followed by descriptions.
11 You have no other skills you can use, and you must exactly follow the
       number of inputs described below.
```

````
The coordinate system is defined relative to the whiteboard, using x and y axes. The x-axis runs horizontally along the whiteboard, while the y-axis runs vertically on it. The origin (0, 0) is located at the lower-left corner of the whiteboard.

Action("draw_line", [x0, y0, x1, y1])
Draw a line on a Whiteboard, x0, y0 being the start point of the line on the whiteboard coordinates, x1, y1 the endpoints.


Additionally, the input to `gen_initial_guess` must be exactly the `initial:DrawState` argument, even if this isn't explicitly used within the function!
The `gen_initial_guess` function MUST return a dict.
If you need to import any modules DO IT INSIDE the `gen_plan` function.
To compensate for an uneven drawing surface and ensure a flat appearance in the top-down camera view, apply 2D offsets to each point, initialized to [0., 0.].
ALWAYS ADD POINT OFFSETS INITIALIZED TO ZERO TO THE `gen_initial_guess`!

Below is one example for a tasks and successful solutions.

# user message
State: DrawState(frames=[])
Goal: Draw a square on the tilted Whiteboard with side lengths of 20cm.

# assistant message
```python
def gen_plan(state: DrawState, pos: list, size: float, offsets: list):
    x, y = pos

    # Define base square corners in order (clockwise)
    base_corners = [
        [x, y],                   # bottom-left
        [x + size, y],            # bottom-right
        [x + size, y + size],     # top-right
        [x, y + size]             # top-left
    ]

    # Add offsets to each corner
    perturbed = [
        [px + dx, py + dy]
        for (px, py), (dx, dy) in zip(base_corners, offsets)
    ]

    # Create draw_line actions
    actions = []
    for i in range(4):
        x0, y0 = perturbed[i]
        x1, y1 = perturbed[(i + 1) % 4]  # Wrap around to close the square
        actions.append(Action("draw_line", [x0, y0, x1, y1]))

    return actions

def gen_initial_guess(initial: DrawState):
    return {
        "pos": [0.32, 0.24], # Center of the drawing board (cm)
        "size": .2, # Square side length (cm)
        "point_offsets": [[0., 0.]] * 4 # 2D point offsets for the square (4 points)
    }
````
````

```
...
```

D.2 Prompt for Push Environment

```

You are a franka panda robot operating in an environment with the
    following state:

TABLE_BOUNDS = [[-0.5, 0.5], [-0.5, 0.5], [0, 0]]  # X Y Z
TABLE_CENTER = [0, 0, 0]

@dataclass
class Frame:
    name: str
    x_pos: float
    y_pos: float
    z_pos: float
    x_rot: float
    y_rot: float
    z_rot: float
    size: float | list[float]
    color: list[float]


@dataclass
class PushState(State):
    frames: List[Frame] = field(default_factory=list)

    def getFrame(self, name: str) -> Frame:
        for f in self.frames:
            if f.name == name:
                return f
        return None


You have access to the following set of skills expressed as pddl
    predicates followed by descriptions.
You have no other skills you can use, and you must exactly follow the
    number of inputs described below.
The coordinate axes are x, y, z where x is left/right from the robot
    base, y the distance from the robot base, and z is the height off
    the table.

Action("pick", [frame_name], pick_axis)
Grab the frame with name frame_name. Aligns the gripper x-axis (the
    axis in which the fingers move) with the pick_axis, if set to None
    , the all axis are checked and which ever one is feasible gets
    used.

Action("place_sr", [x, y, z, rotated, yaw])
Place grasped object at pose x, y, z, with a specific yaw angle. If
    the rotated boolean is set to True, it will rotate the block 90
    degrees around the pick axis. The yaw, which is in radians,
    determines the angle at which the object is rotated with respect
    to the object's current local axis pointing upwards. If z is set
    to None, the object gets plazed on the table. If yaw is set to
    None, there are no restrictions to the yaw angle.

Action("push_motion", [start_x, start_y, end_x, end_y])
Perform a push motion along the straight 2D path defined by the start
    and end points.

Your goal is to generate two things:

```

````
First, generate a python function named `gen_plan` that can take any continuous inputs. No list inputs are allowed.
and return the entire plan with all steps included where the parameters to the plan depend on the inputs.

Second, generate a python function `gen_initial_guess` that returns a set of initial guesses for the continuous input parameters. The number of initial guesses in the
`gen_initial_guess` should exactly match the number of inputs to the function excluding the state input.

The main function should be named EXACTLY `gen_plan` and the initial_guess of the main function should be named EXACTLY `gen_initial_guess`. Do not change the names. Do not create any additional classes or overwrite any existing ones.
Aside from the inital state all inputs to the `gen_plan` function MUST NOT be of type List or Dict. List and Dict inputs to `gen_plan` are not allowed.

Additionally, the input to `gen_initial_guess` must be exactly the `initial:PushState` argument, even if this isn't explicitly used within the function!

#define user
initial=PushState(frames=[Frame(name="block_red", x_pos=0.0, y_pos=0.0, z_pos=0.71, x_rot=-0.0, y_rot=0.0, z_rot=-0.0, size=[0.04, 0.04, 0.12, 0.0], color="[255, 0, 0]"), Frame(name="block_green", x_pos=0.15, y_pos=0.0, z_pos=0.71, x_rot=-0.0, y_rot=0.0, z_rot=-0.0, size=[0.04, 0.04, 0.12, 0.0], color="[0, 255, 0]"), Frame(name="block_blue", x_pos=0.3, y_pos=0.0, z_pos=0.71, x_rot=-0.0, y_rot=0.0, z_rot=-0.0, size=[0.04, 0.04, 0.12, 0.0], color="[0, 0, 255]"), Frame(name="l_gripper", x_pos=0.0, y_pos=0.28, z_pos=1.27, x_rot=0.5, y_rot=-0.0, z_rot=1.29, size=[0.03], color="[229, 229, 229]"), Frame(name="table", x_pos=0.0, y_pos=0.0, z_pos=0.6, x_rot=-0.0, y_rot=0.0, z_rot=-0.0, size=[1.0, 1.0, 0.1, 0.02], color="[76, 76, 76]")])
Goal: Build a bridge. A bridge is defined as two vertical blocks next to each other and one horizontal block on top of them.

#define assistant
```python
def gen_plan(state: PushState, center_x: float, center_y: float, yaw: float, slack: float):

    import numpy as np

    # Build the bridge
    actions = []

    block_size_z = state.getFrame("block_red").size[2]

    # Red #
    pos_x = np.cos(yaw) * block_size_z * .5 + center_x
    pos_y = -np.sin(yaw) * block_size_z * .5 + center_y
    actions.append(Action("pick", ["block_red", None]))
    actions.append(Action("place_sr", [pos_x, pos_y, None, None, None]))

    # Green #
    pos_x = -np.cos(yaw) * block_size_z * .5 + center_x
    pos_y = np.sin(yaw) * block_size_z * .5 + center_y
    actions.append(Action("pick", ["block_green", None]))
    actions.append(Action("place_sr", [pos_x, pos_y, None, None, None]))

````
````

```
    # Blue #
    pos_z = block_size_z + slack
    actions.append(Action("pick", ["block_blue", None]))
    actions.append(Action("place_sr", [center_x, center_y, pos_z, True, yaw]))

    return actions

def gen_initial_guess(initial:PushState):
    guess = {
        "center_x": .2, # BBO initial value
        "center_y": .2,
        "yaw": .0,
        "slack": .03,
    }
    return guess
```

```
#define user
initial=PushState(frames=[Frame(name="big_red_block", x_pos=-0.2, y_pos=0.3, z_pos=0.7, x_rot=-0.0, y_rot=-0.0, z_rot=1.57, size=[0.1, 0.2, 0.1, 0.0], color="[204, 51, 63]"), Frame(name="target_pose", x_pos=0.4, y_pos=0.3, z_pos=0.7, x_rot=-0.0, y_rot=0.0, z_rot=-2.51, size=[0.1, 0.2, 0.1, 0.0], color="[0, 255, 0]")])
Goal: Push the red block to the taget pose.

#define assistant
```python
def gen_plan(state: PushState,
             a_start_x_offset: float, a_start_y_offset: float,
             a_end_x_offset:   float, a_end_y_offset:   float,
             b_start_x_offset: float, b_start_y_offset: float,
             b_end_x_offset:   float, b_end_y_offset:   float):

    import numpy as np

    # Build the block towards the target
    actions = []
    red_box = state.getFrame("big_red_block")
    target = state.getFrame("target_pose")
    dir = np.array([target.x_pos, target.y_pos]) - np.array([red_box.x_pos, red_box.y_pos])
    dir_normed = dir / np.linalg.norm(dir)

    # First push start
    offset_mag = max(red_box.size[:2]) * 3
    a_start_x = red_box.x_pos - dir_normed[0]*offset_mag + a_start_x_offset
    a_start_y = red_box.y_pos - dir_normed[1]*offset_mag + a_start_y_offset

    # First push end
    a_end_x = target.x_pos + a_end_x_offset
    a_end_y = target.x_pos + a_end_y_offset

    # Second push start
    b_start_x = target.x_pos - dir_normed[0]*.2 + a_end_x_offset
    b_start_y = target.y_pos - dir_normed[1]*.2 + a_end_y_offset

    # Second push end
    b_end_x = target.x_pos + a_end_x_offset
    b_end_y = target.x_pos + a_end_y_offset

    # First Push #
```
```

```
    actions.append(Action("push_motion", [a_start_x, a_start_y,
    a_end_x, a_end_y]))

    # Second Push (For adjusting position) #
    actions.append(Action("push_motion", [b_start_x, b_start_y,
    b_end_x, b_end_y]))

    return actions

def gen_initial_guess(initial: PushState):
    return {
        "a_start_x_offset": .0, # BBO initial value
        "a_start_y_offset": .0,
        "a_end_x_offset": .0,
        "a_end_y_offset": .0,
        "b_start_x_offset": .0,
        "b_start_y_offset": .0,
        "b_end_x_offset": .0,
        "b_end_y_offset": .0,
    }
```