Physically Grounded Optimal Realizations of Symbolic Plans

Andreu Matoses Gimenez, Nils Wilde, Chris Pek, Javier Alonso-Mora
Department for Cognitive Robotics, Delft University of Technology
A.MatosesGimenez@tudelft.nl

***Abstract*—Robot autonomy often involves planning high-level discrete decisions and continuous motion planning to realize each decision. Task and Motion Planning (TAMP) algorithms solve these hybrid problems jointly while considering constraints between the discrete symbolic actions, i.e., the task plan, and their continuous geometric realization. Previous TAMP algorithms have mostly focused on computational performance, completeness, or optimality. However, due to the required simplifications and abstractions, the resulting plans often do not account for robot dynamics, nor complex contacts. They also often ignore the effect of the low-level controllers on the optimality and/or feasibility of the plan's realizations. This work investigates the use of a parallelized physics simulator to compute realizations of the plan with a motion controller, realistic dynamics, and considering contacts with the environment. Using cross-entropy optimization, we sample the parameters used by the controllers, or actions, to obtain low-cost solutions. The resulting realized plan is straightforward to implement in the real system, as the robot uses the same controllers. We test our approach for a pick and place task, where our method is capable of finding low-cost feasible solutions in 1-2 min.**

I. Introduction & Related Work

Solving task planning problems that accomplish high-level symbolic specifications in real robot settings requires autonomously deciding both what to do, i.e., the sequence of high-level actions, and how to do it, i.e., the associated motions. This problem is often complex as early actions and the way they are applied may significantly impact later actions' performance or even feasibility. TAMP is an approach to solve simultaneously the symbolic (i.e., actions) and the geometric (i.e., motion) constraints on the solution [1].

Solving TAMP problems is computationally expensive. Evaluating a symbolic plan candidate (i.e., a sequence of symbolic actions) requires finding the appropriate action parameters that make the generated motions optimal or at least feasible. TAMP problems then need to consider the combinatorially large set of possible symbolic plans that accomplish the high-level specification, as well as the sets of possible action parameters. As a consequence, most real-world scenarios require TAMP algorithms to be computationally efficient and to often use simplifications and heuristics.

Existing TAMP solvers divide the problem into hierarchical feasibility (and optimality) stages with progressively more detail to filter the space of symbolic plans. Optimization-based approaches [2, 3] use tree search over sequences of symbolic actions, and evaluate the feasibility and optimality of each node with increasing detail as heuristic. The computation of the detailed motion relies on carefully crafted mathematical constraints and kinematics models that can be efficiently used in specialized optimizers [4, 5], which, depending on the problem formulation, may not find a solution.

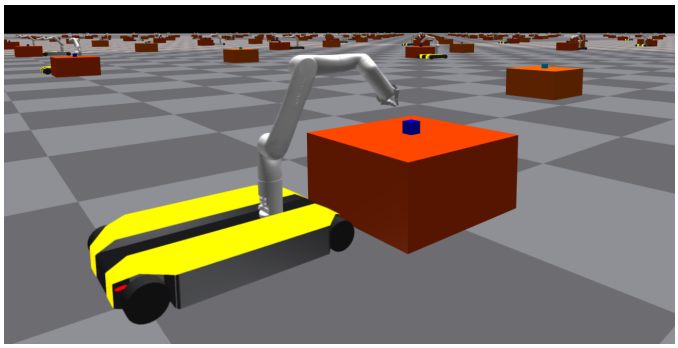

Fig. 1: Parallel instances for different realizations of the same plan, where the robot is performing a grasping motion in the Isaac Gym physics simulator.

In [6, 7], sampling based approaches are used in combination with Planning Domain Definition Language (PDDL) [8] symbolic planners to incrementally generate PDDL subproblems from discretized action parameters. Thomason et al. [9] introduces an optimal sampling-based approach using asymmetric bidirectional sampling, which significantly improves the efficiency of finding action parameters for plans without the need for discretization. These methods require the creation of samplers that generate constraint-satisfying configurations of the robot and objects, for the predicates in preconditions and effects of the actions operators. The samplers need to be efficient, and therefore, encoding complex dynamics and contacts in the predicates is often not feasible.

This work investigates the use of GPU-based physics simulators to efficiently find high-quality realizations of a plan. The realization of the actions accounts for the effects of the motion controllers that the real robot will use, as well as complex contacts due to the robot and environment geometry, Figure 1. As the complexity of modeling contacts and dynamics is offloaded to the simulator, the complexity and number of predicates, preconditions, and effects of the actions are significantly reduced. Moreover, we use efficient and parallelizable low-level controllers to perform the actions, which allows us to only sample the parameters that define the resulting motion instead of sampling the motion itself.

This has the additional benefit of generating plans that are straightforward to implement in the real system and will use the same controllers. The parameters are sampled following a probability distribution, which is iteratively brought closer to the optimal distribution using cross-entropy optimization.

II. Problem Formulation

We consider robotic task planning problems, where a sequence of actions needs to be completed successfully to achieve a final symbolic specification ψ_g while minimizing a cost. In addition, we consider dynamical properties of the system and complex contacts. Previous actions may affect subsequent actions' feasibility and cost.

The configuration space of the system is $\mathcal{X} = \mathcal{Q} \times SE(3)^m$, where $\mathcal{Q}$ denotes the configuration space of the robot, and there are m objects in the environment. Let $\mathbf{x}_t \in \mathcal{X}$ be the state of the whole system at time t. The final specification is accomplished if $\psi_g(\mathbf{x}_T) = True$ for the final time T

We define symbolic parameters $\mathcal{Z} = \{z_1, \ldots, z_k\}$ representing variables of interest of the problem. As an example, consider a $SE(3)$ grasp pose required to grasp a block; we call this symbolic parameter $graspCube$. The values that a parameter can take are constrained by $h(z_j) \leq 0$, which will be referred to as the region of the parameter. For $graspCube$, this could mean that we consider all grasp poses whose locations are on a spherical surface around the cube.

The robot can execute a finite set of action templates $\mathcal{A} = \{\alpha_1, \ldots, \alpha_w\}$. which can be considered an abstraction of a specific high-level robot skill (e.g., grasping, placing, moving, etc.). We use the connotation "template" as several actions can be generated from the same template by instantiating them with different symbolic parameters. Action templates are defined by the tuple

$$\alpha_i = (\mathcal{Z}_{\alpha_i},\ \text{controller}_{\alpha_i},\ c_{\alpha_i}(x_t),\ \text{success}_{\alpha_i}), \tag{1}$$

where $\mathcal{Z}_{\alpha_i}$ is a set of parameters compatible with the action template. A parameter is compatible if it is consistent with the controller's type of input. For example, if the action template uses a PID point $\in \mathbb{R}^2$ tracking controller, the type of symbolic parameter must be a $\mathbb{R}^2$ point (within a region).

The $\text{controller}_{\alpha_i}$ uses the parameters to calculate the action motion and control the robot. The action template has a cost $c_{\alpha_i}(\mathbf{x}_t)$, which can be continuously evaluated or only once when the action is finished (terminal cost). The success of the action can be determined by evaluating the boolean predicate $\text{success}_{\alpha_i} \in \{True, False\}$, which depends on the generated motion during the action execution.

When a compatible assignment of symbolic parameters $Z_{a_i} \subseteq \mathcal{Z}$ is made to instantiate an action template, we obtain a symbolic action a_i. As an example, consider the following symbolic actions resulting from a different choice of parameters for the template $move\_to$: $move\_to(Exit, \ldots)$, $move\_to(Table, \ldots)$. We say that the action is "realized" when a value is given to its symbolic parameters Z_{a_i}. A realized action may not necessarily be successful.

A symbolic plan is defined as a sequence of symbolic actions $\pi = \{a_1, \ldots, a_n\}$ and its symbolic parameters are the parameters of its actions, $Z_\pi = [Z_{a_1}, \ldots, Z_{a_n}]$. A plan is realized when all its actions are realized, and it is feasible if all actions can be executed and the goal specification is achieved. The total cost of a realized plan is the sum of all realized action costs plus a final cost.

$$c(\pi) = c_{end}(\mathbf{x}_{0:T}) + \sum_{i=1}^{n} \sum_{t=t_{0,i}}^{T_i} c_{a_i}(\mathbf{x}_t), \tag{2}$$

where $t_{0,i}$ and T_i represent the start and end time of a_i.

Problem 1 (Symbolic Plan Realization)**.** Given a sequence of symbolic actions $\pi = \{a_1, \ldots, a_n\}$, find a realization Z_π that solves

$$\begin{aligned} \min_{Z_\pi} \quad & c(\pi) \\ \text{s.t.} \quad & \text{success}_{a_i} = True,\ \forall\ a_i \in \pi \\ & \psi_g(\mathbf{x}_T) = True \end{aligned} \tag{3}$$

In Problem 1 we assume that for a given action sequence, there exists a realization of the parameters of each action Z_π^* that makes the plan feasible.

III. Method

In this section, we outline the methods used to evaluate plan realizations with the required level of detail in a reasonable time. To accomplish this, we use parallelized implementations of the physics simulator and low-level controllers to evaluate plan realizations. Finally, we use a sampling based optimization approach to find the parameters of the pan.

A. Simulation of the Dynamics

To accurately model the dynamics and complex contacts of the robot interacting with the objects in the environment, we use the GPU-based physics simulator IsaacGym [10]. The simulator allows for computationally efficient parallelized execution of plans, which makes sampling-based methods feasible, see Figure 1.

B. Low Level Controllers

1) Optimization Fabrics: Used for fast computation of full body motions (mobile base + manipulator) when end effector pose control is required. The controller is based on Riemannian Motion Policies (RMPs) [11] with convergence and safety guarantees. The system is forced by the potential ψ to the desired minimum with attractor weight γ and damping $\boldsymbol{B}$.

$$\tilde{\boldsymbol{M}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\ddot{\boldsymbol{q}} + \tilde{\boldsymbol{f}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \gamma \partial_{\boldsymbol{q}} \psi + \boldsymbol{B}\dot{\boldsymbol{q}} = 0 \tag{4}$$

Solving (4) yields the trajectory generation policy in acceleration form $\ddot{\boldsymbol{q}} = \tilde{\pi}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, which is known as static fabrics [12]. The equation can be modified to account for dynamic obstacles such as moving obstacles and reference path tracking [13].

2) Path Finding: A* [14] is used to compute collision-free paths for the mobile base. The generated waypoints are tracked with a PID controller. A* is computationally efficient for low-dimensional problems with appropriate discretization, making it suitable for our use case.

Algorithm 1 Plan Parameter Cross-Entropy Optimization

Initialization:

$\mathcal{Z}_\pi \leftarrow$ Symbolic parameters used by the actions
$\phi(\theta_\pi^0) \leftarrow$ Parameterized initial distribution
$n_{env} \leftarrow$ Number of parallel plans
$N_e \leftarrow$ Number of important samples
$j = 0$ ▷ CE iteration counter
while θ_π^j not converged **do**
  $S \leftarrow$ Sample n_{env} times Z_π from distribution $\phi(\theta_\pi^j)$
  $C \leftarrow$ Evaluate successful plan costs of samples in S
  $\theta_\pi^{j+1} \leftarrow$ Update distribution based on top N_e samples
  $j = j + 1$
end while
return Best sample $Z_\pi \in S$, and final θ_π^{j+1}

C. Cross-Entropy Optimization of Parameters

In order to optimize the parameters Z_π of a given plan π, we use Cross-Entropy optimization (CE) [15]. The method iteratively refines the estimation of the optimal parameters by using a form of importance sampling and updating the parameters based on the samples that yield better results.

We assign a parametrized probability distribution of appropriate dimension to each parameter of an action $\mathbf{z}_k \sim \phi(\theta_k) \ \forall \ \mathbf{z}_k \in Z_{a_i}$. Let $\phi(\theta_\pi)$ be the resulting probability distribution of all the parameters of a plan π. For convenience, we refer to sampling all the parameters of the plan as sampling the vector $Z_\pi \sim \phi(\theta_\pi)$. The outline of the CE optimization used can be seen in Algorithm 1

IV. Preliminary Results

We test the method for a pick and place task with a mobile manipulator, see Dingo-O with Kinova 6DoF arm and gripper in Figure 1. The robot needs to grasp a block from a table, place it on another table, and move to the exit. We define the following action templates:

$$move\_to(\{loc\}, \text{A*}, c_m(x_t), in(loc)) \tag{5}$$
$$grasp(\{g_p\}, \text{Fabrics}, c_g(x_t), ee\_in(g_p)) \tag{6}$$
$$place(\{g_p\}, \text{Fabrics}, c_p(x_t), ee\_in(g_p)) \tag{7}$$

The A* implementation in (5) uses a 2D discretized version of the environment, avoiding obstacles (the tables). The waypoints are then tracked by a PID controller, which controls the velocity of the mobile base. During $move\_to$ The arm remains static. The cost c_m is 1 every time step the action is not finished. The success condition is true if the mobile base is at the goal location within a tolerance $in(loc) : \|x_{base} - x_{loc}\| \leq \epsilon$

For (6) and (7), we use as input parameters a grasp pose g_p and a full body (mobile base + arm) parallelized fabrics controller. As before, The cost is simply 1 every time step the action is not finished. The success condition is true if the end effector has reached the grasp/place pose within a tolerance $ee\_in(g_p) : \|g_p - p_{ee}\| \leq \epsilon$

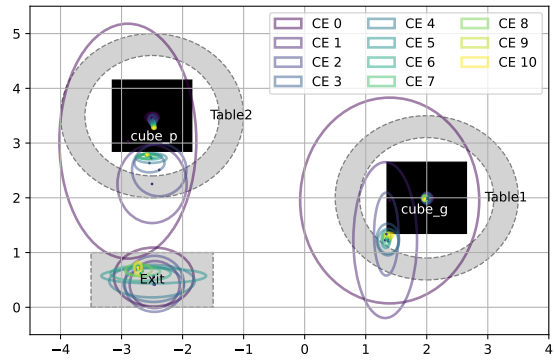

Fig. 3: Evolution of the distribution of the parameters with each CE iteration. For 3D parameters, only the first two dimensions are shown.

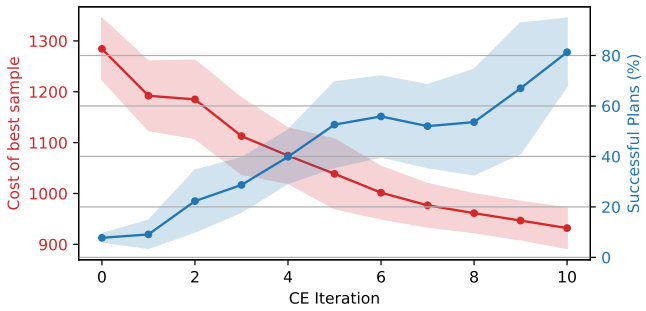

Fig. 4: Evolution of the cost of the best-sampled plan (left axis). Change in percentage of sampled plans that are evaluated as successful (right axis). Averaged over 10 runs, filled area represents $\pm\sigma$.

We define the following symbolic parameters: $Table_1$ and $Table_2$, annular regions of $\mathbb{R}^2$ points around the table with the block and around the table where the block can be placed, respectively. $Cube_g$ and $Cube_p$ are hemisphere surface regions around the block and around the center of the second table, respectively. Finally, $Exit$ is a rectangular region of $\mathbb{R}^2$ points centered around the robot's resting area.

We create a candidate symbolic plan by extending the current action templates with appropriate preconditions and effects, and using an off-the-shelf PDDL solver [16]. The resulting plan accomplishes, at least at the symbolic level, the goal of moving the block to the second table, and ends with the robot in the designated exit area.

$$\pi = move\_to(Table_1), grasp(Cube_g), \\ move\_to(Table_2), place(Cube_p), move\_to(Exit) \tag{8}$$

We evaluate a realized plan as successful if the grasp and place actions were successful, and if the robot's mobile base final position is inside the region $Exit$. Notice how the requirement of a successful plan does not enforce the robot to be in the annular regions $Table_1, Table_2$. Sampling over these regions serves as an initial guess of the distribution.

We run the CE Algorithm 1, by initially sampling values uniformly over the whole region of the parameters. For subsequent iterations, we model the distribution of the parameters

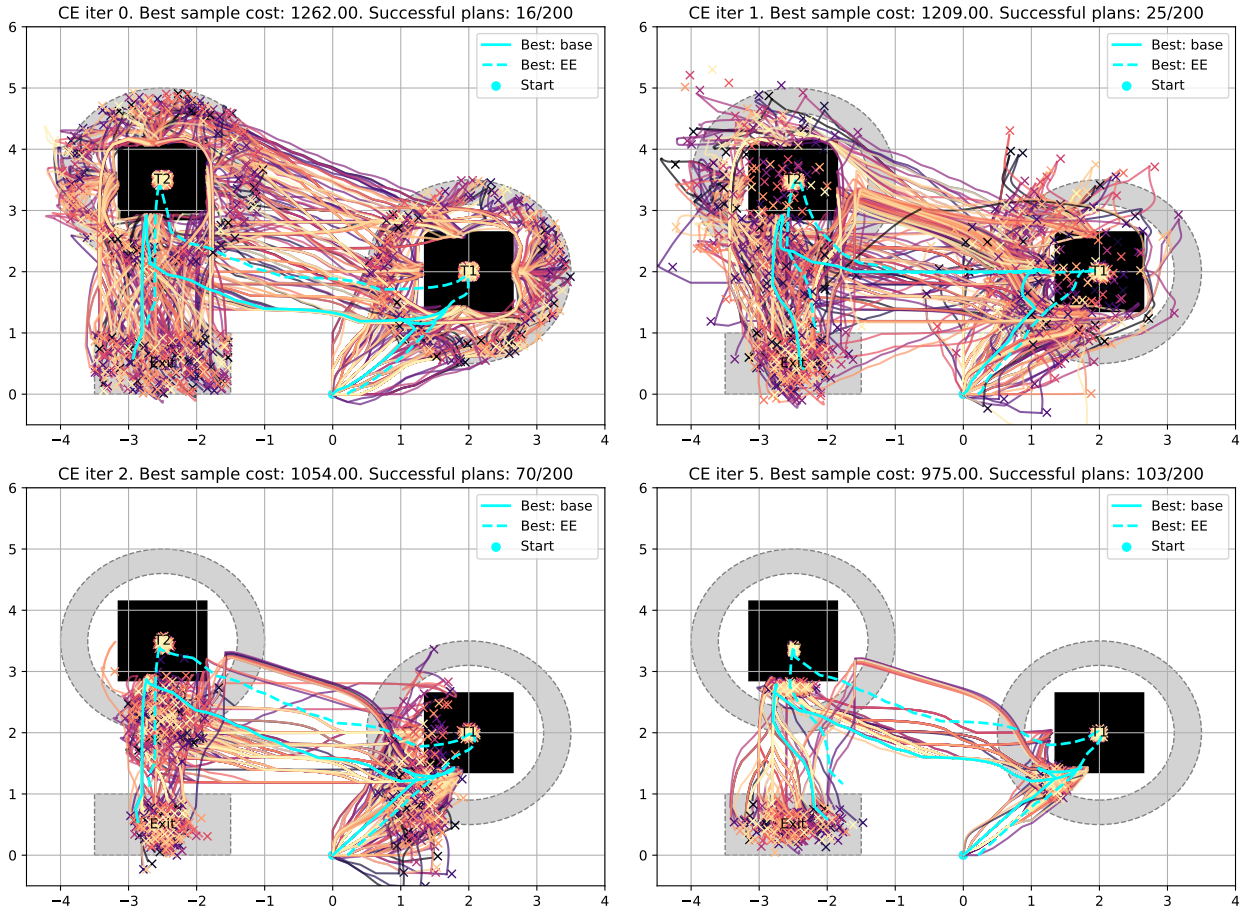

Fig. 2: Plans simulated from the sampled parameters for different iterations of the CE optimization. The trajectories displayed in orange tones show the trajectory of the mobile base. The cross markers show the value of the parameter. The best-sampled plan is shown in cyan, both base and end effector trajectory (EE), in continuous and dashed lines, respectively. The parameter areas are displayed in clear grey. The tables are in black, with 20cm of additional padding, as used by the A* algorithm.

as an independent normal distribution. The resulting μ and σ of the elite parameter samples are used for the next iteration.

We use $n_{envs} = 200$ and $N_e = 10$, and perform 10 CE iterations. The resulting sampled plans at different stages can be seen in Figure 2, with the sampled parameters and resulting trajectories projected into the ground plane. The end effector trajectory is shown only for the best sample of the CE iteration.

Figure 3 shows the parameters' normal distributions as they evolve with each iteration. The 2D distribution is shown as an ellipse with the corresponding semi-axis being 2σ. For the 3D pose parameters, only x, y is shown.

Only $\approx 10\%$ of the sampled plans are successful during the initial iteration with uniform distribution. This is mostly due to the unlikeliness of sampling a grasp pose reachable from the random position around the table. As the iterations progress, the successful plan samples significantly increase, and the cost of the best sampled plans decreases Figure 4.

V. Conclusion and Next Steps

The preliminary results showcase that our method is capable of finding low-cost realizations of a given plan within few CE iterations. The solution takes into account the geometrical constraints of the robot, such as the inability to reach objects from certain positions. Our formulation does not require one to explicitly model these constraints in the problem definition, and instead, they appear as a consequence of the simulation. The effects of the specific controller can be seen when the mobile base is repositioned while pushing against the table during grasping and placing. This behavior resulting from contacts and controllers could not be accounted for in previous sampling-based methods [7, 9]

The results also show that our method requires further speed up in order to be used in a real setting ($\approx 30s$ per CE iteration), where replanning is needed to react to unexpected changes in the environment within a reasonable time, especially when different plans need to be compared. Our next step is to introduce the search over different sequences of actions in the optimization problem [17]. By taking advantage of the low computational cost of adding parallel plan evaluation, we expect a significant speed-up . Moreover, we will explore breaking plans into smaller action sequences that can be simulated in parallel and stitched together.

ACKNOWLEDGMENTS

This project has received funding from the European Union through ERC, INTERACT, under Grant 101041863. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

[1] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated Task and Motion Planning, October 2020. URL http://arxiv.org/abs/2010.01083. arXiv:2010.01083 [cs].

[2] Marc Toussaint. Logic-geometric programming: an optimization-based approach to combined task and motion planning. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 1930–1936, Buenos Aires, Argentina, July 2015. AAAI Press. ISBN 978-1-57735-738-4.

[3] Marc Toussaint and Manuel Lopes. Multi-bound tree search for logic-geometric programming in cooperative manipulation domains. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4044–4051, May 2017. doi: 10.1109/ICRA.2017.7989464. URL https://ieeexplore.ieee.org/document/7989464.

[4] Marc Toussaint. Newton methods for k-order Markov Constrained Motion Problems, July 2014. URL http://arxiv.org/abs/1407.0414. arXiv:1407.0414 [cs].

[5] Rachel Holladay, Tomas Lozano-Perez, and Alberto Rodriguez. Force-and-Motion Constrained Planning for Tool Use. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7409–7416, Macau, China, November 2019. IEEE. ISBN 978-1-72814-004-9. doi: 10.1109/IROS40897.2019.8967889. URL https://ieeexplore.ieee.org/document/8967889/.

[6] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Sampling-based methods for factored task and motion planning. *The International Journal of Robotics Research*, 37 (13-14):1796–1825, December 2018. ISSN 0278-3649. doi: 10.1177/0278364918802962. URL https://doi.org/10.1177/0278364918802962. Publisher: SAGE Publications Ltd STM.

[7] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning, March 2020. URL http://arxiv.org/abs/1802.08705. arXiv:1802.08705 [cs].

[8] D. McDermott, M. Ghallab, A. Howe, Craig A. Knoblock, A. Ram, M. Veloso, Daniel S. Weld, and D. Wilkins. PDDL-the planning domain definition language. 1998. URL https://planning.wiki/_citedpapers/pddl1998.pdf.

[9] Wil Thomason, Marlin P. Strub, and Jonathan D. Gammell. Task and Motion Informed Trees (TMIT*): Almost-Surely Asymptotically Optimal Integrated Task and Motion Planning. *IEEE Robotics and Automation Letters*, 7(4):11370–11377, October 2022. ISSN 2377-3766, 2377-3774. doi: 10.1109/LRA.2022.3199676. URL https://ieeexplore.ieee.org/document/9869707/.

[10] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.

[11] Nathan D. Ratliff, Jan Issac, Daniel Kappler, Stan Birchfield, and Dieter Fox. Riemannian motion policies, 2018.

[12] Nathan D. Ratliff, Karl Van Wyk, Mandy Xie, Anqi Li, and Muhammad Asif Rana. Optimization Fabrics, August 2020. URL http://arxiv.org/abs/2008.02399. arXiv:2008.02399 [cs, math].

[13] Max Spahn, Martijn Wisse, and Javier Alonso-Mora. Dynamic Optimization Fabrics for Motion Generation, March 2023. URL http://arxiv.org/abs/2205.08454. arXiv:2205.08454 [cs].

[14] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/tssc.1968.300136. URL https://doi.org/10.1109/tssc.1968.300136.

[15] Zdravko I. Botev, Dirk P. Kroese, Reuven Y. Rubinstein, and Pierre L'Ecuyer. Chapter 3 - The Cross-Entropy Method for Optimization. In C. R. Rao and Venu Govindaraju, editors, *Handbook of Statistics*, volume 31 of *Handbook of Statistics*, pages 35–59. Elsevier, January 2013. doi: 10.1016/B978-0-444-53859-8.00003-5. URL https://www.sciencedirect.com/science/article/pii/B9780444538598000035.

[16] M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, July 2006. ISSN 1076-9757. doi: 10.1613/jair.1705. URL http://dx.doi.org/10.1613/jair.1705.

[17] Reuven Rubinstein. The Cross-Entropy Method for Combinatorial and Continuous Optimization. *Methodology And Computing In Applied Probability*, 1(2):127–190, September 1999. ISSN 1573-7713. doi: 10.1023/A:1010091220143. URL https://doi.org/10.1023/A:1010091220143.