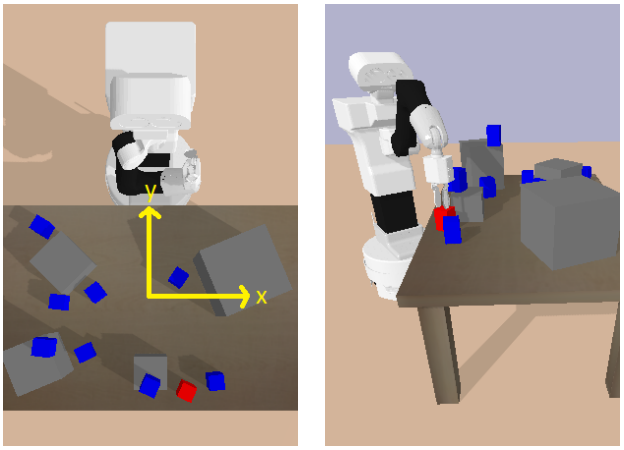
Deep object-centered Heuristic for Robot Planning

Dominik Marino*, Tianyu Ren*
*Computer Science Department, Technische Universitat Darmstadt

***Abstract*— Task and Motion Planning (TAMP) requires a high load of computation through exploiting environment knowledge and selecting appropriate robot actions to integrate feasible motion planning. Hierarchical structures with symbolic placeholder have a fundamental problem with generating a variety of feasible skeleton plans in prompt evaluation time. The lack of performance in diversity TAMP algorithm can lead to a dead end due to missing information in skeleton space. A structural planning algorithm based on action costs for decision-making and extending the space of skeleton plan can efficiently combine the balance between exploration-exploitation at each decision node in the planning progress due to low-cost solution for only evaluating feasible skeleton plans. Monte-Carlo Tree Search (MCTS) ensures a single extended decision space to optimize minimum cost solutions in task domains for exploiting existing candidate plans. This enables a powerful planning algorithm for long-horizon planning tasks. In this paper, we empirically evaluate our algorithm on different robotic planning environments that requires dead-ends, or setting different difficulties in the domain space. We will show, that using deep learning methods can lead to better performance and increases the success of alternating extending skeleton plan. Our approach will be compared to a naive random based decision plan with several trained neural networks in the decision-making search.**



(a) Initialization of environment (b) Found skeleton plan

Fig. 1. Example Task: Using a simple skeleton plan to pick a target body $body1$ on a region $region1$. Fig. 1(a) shows an example of an individual environment space, where a robot has to search a configuration without any collision. Fig. 1(b) is a successfully found configuration space in the environment. The environment has an global coordinate system and for all object, we use the global coordinate system for the grasping direction

I. INTRODUCTION

A central problem in robot planning, is the variety of solutions for a specific tasks. A robot should perform on multiple task in unknown habitats. This can be in hospitals, museum or a company, where an intelligent robot has to act in different environments. This requires a skill-set in various disciplines in *Task and Motion Planning (TAMP)*. Multiple decisions and a reliable plan has to be prepared to reach the target object [1], [2].

A solution would be a planner for different approaches to operate chronological sequence and manipulate in an environment [3]. However, a task can be separated into several subsequences. By breaking down a complex problem into multiple subparts to get a reliable task and motion planning algorithm that is more tractable, w.r.t. the properties of the environment [2], [4], [5], [6], [7]. Intelligent robots have to perform in every scenario and a task planner could decide long-term strategies, whether a robot may want to discover a new position, like, the target object is not in the workspace or a target object is in the workspace but not graspable on the current position [3], [4]. Therefore, individual decisions have to be considered to guarantee a successful planning in multibody robots. Planning trajectories is an important objective of intelligent robots, but a difficult task to accomplish [8].

This can lead to difficult challenges for searching a solution, for example, to a desired position, which a target object can be grasped from or finding a configuration of all joints. The problem get more complex, when it comes to moving objects. Common approaches need a lot of computation time, up to minutes, to navigate in a complex environment [9], [10].

By getting more advanced in planning algorithm, with *Artificial Intelligence (AI)* planning or different hierarchical structures approaches [11], TAMP generate actions over the current state to reach the goal state [1]. In most cases, hierarchical structures do not have enough domain knowledge. Generating knowledge is one of the fundamental jobs to produce correct results in the domain space [3].

A high-level skeleton plan can be easily separated into multiple subsequences, where every state in the graph structure tries to solve one of the subtasks. Sometimes it is necessary to have a more detailed generalization of the domain space to make better assumptions in the planning process. But this is inapplicable to most real-world scenarios [9]. To integrate this approach, all the obstacles in the environment are already observed, and this can be taken, with laser sensors or cameras to discover all positions and orientations of the bodies in the world environment. This, in particular, makes it easier for a hierarchical TAMP structure for solving long-horizon tasks [3].

The main approach is to generate sequences with minimal assumptions for each state in a skeleton plan. The simplest implementation in a TAMP planner is a random based decision. The goal of this experiment is to reach a target object. Fig. 1(a) shows a simple environment, where geometric object will be placed on a region, and the planner have to made multiple decisions, like, is the geometric object is reachable in the environment. A heuristic task planner is used to update the states in each node for representing knowledge by a formalism language to solving the planning problem [12]. A feasible plan will be generated for each individual layer in the interface. The indicated random decision nodes are an inefficient way for implementing such a discrete planning specification and will later be replaced with a *Convolutional Neural Network (CNN)* for predicting accessible directions of the geometric targets. A *Neuronal Network (NN)* evaluates the current position w. r. t. the grasp direction that is given from the CNN. The idea of the NN is to improve possible base positions that will lead to failure in later planning progress.

TAMP should generate multiple skeleton plans by optimizing the state segments by using action parameters. This should guarantee, that there is no better solution for this skeleton plan [3], [13]. In combination with PDDLStream, all decisions are implemented as a stream for each state, by using symbolic constraints for sampling alternative skeletons. *Monte-Carlo Tree Search (MCTS)* is a general approach to reduce the probability of exploring bad behaviors and avoid complete moves in different disciplines [14]. In combination with a heuristic TAMP planner, it will update the value properties in the streams for the skeleton when it's needed [3].

II. RELATED WORK

In recent years, planning is a big research area in robotics application. The property of an adequate planning algorithm is to create feasible approaches to generate robust skeleton plan in an unexpected environment specification [3], [9], [11]. In particular, these methods can be described in path planning, where the purpose of sampling configurations and automatic execute a successfully found TAMP using a robot without any collisions [6], [15].

Latest studies exhibited by using TAMP methods as an optimization problem, by exploring the search space for solving real-world TAMP problems have some issues with symbolic guidelines. Unknown environment knowledge for a TAMP method obligates a symbolic plan, that can operate infeasible domains [9]. The inflexibility of such methods is the biggest drawback of those methods [2]. However, to search a feasible plan, those approaches ignore the most of the space environment. The result of an incomplete domain leads automatically to performance issue.

In complex environments sequential skeletons are not very suitable anymore and generating alternate skeleton plan is still an open question in the scientific fields. There are several approaches to fit alternating skeleton plans by adding geometric constraints, but the authors Srivastava *et al.* already discussed the problems of rejecting skeleton plans, when the task planner already has captured failure of the current state. In this scenario, it should no longer be possible to configure actions states of the skeleton space [4].

To apply a high-level planner, the author Tianyu *et al.* presents an integrated action planning method for robots with symbolic reasoning, so called eTAMP. So far, their approach has considered arbitrary obstacles, in which a set of initialized random streams and takes decisions that generate a feasible plan when it's found. They addressed a collection of objects in a multiple high-level task planner to apply manipulation, navigation and action planning. It is also intimated a pre-specification like object poses, grasps directions and robot configuration. Using these pre-specifications to generate a heuristic sequence to alternating feasible skeleton plans [3].

However, in this approach they used *Monte-Carlo Tree Search (MCTS)* for expanding the graph structure to ensure exploration-exploitation at each skeleton node. These decision nodes making assumption to maximize expected rewards under the initial state. In an ideal world, it is impossible to discover all possible solutions under a particular circumstance, but to optimize bounded assumptions to make sure to get the best actions for the robot to reach a target object in the environment space. Therefore, a symbolic plan with pre-specification stream makes it a powerful optimization algorithm in long-horizon planning to minimize cost.

III. PRELIMINARIES

To produce a state-space configuration, it is essential to generate a logical plan with discrete symbolic variables that produces output variables. Therefore, a robust notation needs to ensure a heuristic search in intelligent planners. Predefined formulated instruction requires efficient path-planning methods. This should guarantee to maintain the existing structure by changing the individual node pairs with a given configuration.

A. AI Planning

There are a variety of formulations for AI-based planner, this includes the *Planning Domain Definition Language (PDDL)* developed by McDermott and AIPS 1998 as a central language for domain-independent planners. A central role plays in the design of the planner, which affects performance and run-time behaviors [7]. A robot executes a feasible skeleton plan in a certain environment and accomplishes independent tasks, in particular, domains with for example unique properties. An operating plan has to provide suitable notation syntax specified in hierarchies structures as a logic based formulation [11].

PDDL describes a deterministic formal expression of object types, predicates, and actions. The PDDL semantic makes the encoding much smaller and easier to represent the world as a task planner. The robustness of planners and the evaluation of planning issues will be improved by a concrete binding to search action parameters [4], [10], [16].

A PDDL task defines a deterministic task planning problem that can be represented as a tuple $T = \langle O, S, G, A \rangle$, where $Objects$ O describes everything interesting in the world, for example, movable bodies, regions, or trajectories without any collisions. Predicates are the properties of the $Objects$ O that we are interested in. The state can be true or false. The Variable G characterise the $Goal\ specification$ that in the course of execution shall return the status true. An $Action\ operator$ A changes the state of the world [3], [17].

Running a set of actions $a_0, \ldots, a_n$ from an initial state S that generates state sequences $s_1, \ldots, s_{n+1}$ will successful applied to a feasible instantiation which all the predicates and goal specification are true. PDDL defines a fully observed environment to the task, enables planning for scenarios form the initial to the final pose.

PDDLStreams: Many TAMP streams are exceptionally expensive, for example, for calculating inverse kinematics, motion planning and collision checking [7]. For each stream, it generates an instance output as a placeholder and symbolised as a prefix #. This output will be placed as an input to other streams in the skeleton plan. PDDLStream intuitively uses these optimistic outputs values before actually querying any streams to potentially produce real values.

Like in PDDL, a PDDLStream task can be represented as a tuple $T_{stream} = \langle O, S, A, Streams \rangle$. $Streams$ gives the skeleton plan a new output that can be used for the next subtask in the planner. Once a plan uses these optimistic values, the streams actually produce bindings to these placeholders.

This algorithm operates on optimistic objects and then provides a search over the combined set of real objects and optimistic objects. If its find a real plan with only real objects for the skeleton plan, then it return these values otherwise [6]. It samples all the streams that are associated with the objects in the skeleton plan and ensure that on the next iteration a different plan is produced. Optimistic plans may be feasible, but require a substantial amount of rejection sampling and queries only a few samples. Binding algorithm would require many iterations and have to balance computation time spent searching and sampling objects in the state node [6], [18], [17].

PDDLStream is a generic extension of PDDL that supports sampling procedures as blackbox streams [7], [19]. This makes it impossible to prepare a complete plan, and some of these streams serve as a placeholder in on optimistic plan policy π_s. It is about preparation of real task with symbolic constraints in optimistic planning which not necessarily give a feasible plan but essential to decompose planning problems into subtask [19]. A more detailed example of creating an input as a PDDLStream is shown in Table III-A. Here the sample-grasp-direction and sample-grasp is explained more in detailed.

TABLE I

AN EXAMPLE OF A SET OF $Stream_{s_0}$ FOR SAMPLING GRASPS

Stream	Description
$\langle$ sample-grasp-direction($oBody$) $\rightarrow \#voG0\rangle$	Generate a stream for a target body $oBody$ sides an object can w.r.t. geometric object properties be grasp
$\langle$ sample-grasp($oBody, \#voG0$) $\rightarrow \#voG1\rangle$	Generate a end effector frame w. r. t. the measured frame of the object

$oBody$ is the target object to be grasped on a placed region $region1$. Here, a list w.r.t. of the geometric object property orientations is to be returned, which is a possible subset of available grasping directions. With this subset $\#voG0$ and the target object $oBody$, the position and orientation of the end effector robot arm can be determined geometrically in sample-grasp methods and accordingly return the object $\#voG1$.

B. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a powerful approach for solving sequential decision-making problems based on a tree structure [14]. The idea is to find a balance between exploration and exploitation given a probability of actions represented as states, nodes and edges using Monte-Carlo rollouts [20], [21]. Rollouts algorithms based on Monte-Carlo control estimates actions values for a policy by averaging the return by simulating possible action and follow the given policy π_i [14], [22], [23]. When the estimation is accurate enough by simulating a trajectory, the process rollouts results to next state. The difference between Monte-Carlo estimate and a rollout policy is, that in Monte-Carlo estimate it approximate a complete optimal action-value function for a given policy [14]. Rollout produces Monte-Carlo estimates of action values only for each current state for a given policy. This makes rollout simple algorithm in decision-time planning by immediately discard action-value estimation for every state-action pair outcome [21], [24]. The improvement of rollout algorithm, for example, in MCTS algorithm takes the advantage of using the property of the policy w.r.t. estimated action values[25].

MCTS learn which action has to be taken in each state and maximizes the expected reward. The goal is to learn a policy $\pi : \Omega \rightarrow A$, where $\pi(s)$ chooses the best action for a state s_{t+1} to maximize the expected reward [26], [27]. The method approximates the immediate action-values function $Q_\pi(s_t, a)$ by exploring different actions. The biggest advantages of MCTS are an efficient approach to improve convergence in heuristic planning [25]. The MCTS algorithm simulates multiple iterations starting at the root node s_t to the current state s_{t+1} by expanding trajectories that have the highest estimation from the simulation more detailed in illustrated in Fig. 2 and consists of the following four steps:

- *Selection*: The algorithm starts at the root state s_t and selecting nodes s_{t+1} level by level. Selection is also called tree policy. It stops when a state of the skeleton is already sampled, but the next node is not fully explored or the current state a leaf node.
- *Expansion*: In the expansion phase, a new node will be added to the tree. The edge represents a decision in the

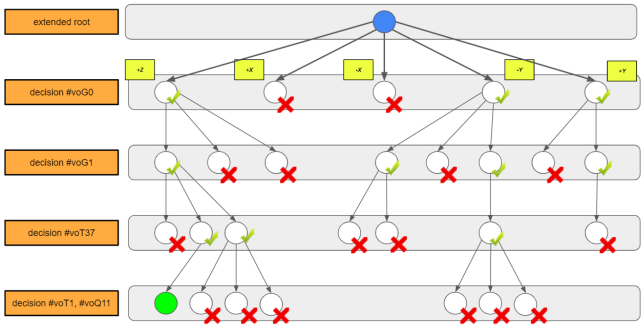


Fig. 2. Representation of a Monte Carlo Tree Search. Execution of many iterations and building incrementally a tree structure from the root node to the current state. Each iteration consists of Selection, Expansion, Simulation and Backup [28]. A tree search is executed on the pre-defined skeleton plan for a simple pick task, and returns individual decisions of each state in the tree graph, based on the learned environment.

tree structure.

- *Simulation*: A random child node will be selected, when the graph structure expands. In this phase, a complete rollout executes until a terminal state is reached.
- *Backup*: Updating values, visits and victories of the node states through the tree from the last reached state in the simulation phase.

MCTS depends on selecting useful action values for its next iteration. However, with rollout, generating an accessible chronological sequence using a simple policy plan [25]. It is possible to create a detailed trajectory that creates a possible skeleton plan and returns a tree with an executable path [25], [29], [30]. For the evaluation of these states and estimate the values of selected actions in the tree policy to find a balance between exploration on unvisited nodes with exploitation of already visited nodes. For this purpose, a tree policy can use ϵ-*greedy* or *Upper Confidence Bound (UCB)* selection rule for selecting actions [28].

C. Upper Confidence Bound for Trees

Using *multi-armed-bandit* problems, a tree policy have to search good behaviors in the tree structure, until a leaf node is reached and found the best sequences in the shortest among of time [24]. To create a tree and take decisions by evaluating feedback from already visited state nodes, it is essential to explore in the tree policy, because of uncertainty about accuracy in action-value estimates [22].

The idea of UCB methods is to achieve exploration by favoring each action step. The balance of exploration-exportation during the search in the environment, by using the knowledge for selecting actions with the highest reward [23], [22]. This principle is based on optimism in fact of uncertainty

$$u = \arg \min_{i \in \{1,\dots K\}} \overline{X}_{i,M_i(n-1)} + C\sqrt{\frac{\log n}{M_i(n-1)}} \quad (1)$$

where u is the decision, $M_i(n) = \sum_{t=1}^{n} \mathbf{1}\{t = i\}$ is the number of times is played up time n. $\overline{X}_{i,M_i(n-1)}$ denotes the average reward of arm i up to time $n-1$. $C = \sqrt{2}$ is an exploration constant.

IV. METHODS

Heuristic search is a classical approach for encounters large trees in state-space planning. In many applications it is required to store the result of planning of a long-horizon task, where the state's focus on planning [3], [28]. Decision-making in a long-horizon planning task is independent of fast responses. Therefore, it is not required to have dozens of plans during task planning, but low latency action selection should be a priority requirement to compute a policy, that can be executed in a short computation time. Usually, doing planning needs continuous updates for each encountered decision state. Fig. 2 shows an example of going through an MCTS tree search and finding a suitable plan that fits for the environment space on a simple skeleton plan referenced in Table V-B.

A. Finding extended decision tree

The MCTS algorithm starts with an extended root node, searching a sequence of decisions until a terminal state is reached. While learning from the environment of past experience to update the sequential decisions in the tree, it will receive rewards $r \in \mathcal{R}$.

$$r = 0.1\left(\frac{depth_{end}}{depth_i} + \frac{1}{motionCost_{end}}\right) + r_{success} \quad (2)$$

where $depth_{end}$ of the current terminated node, and the $motionCost_{end}$ describes the cost of the robot from its initial state to the terminal state. $r_{success} \in \{0, 1\}$ defines a successful binding. The equation (2) excludes already taken decision that are earlier failed in the skeleton plan and elaborate more on branches that takes less cost in robot actions.

MCTS π_i binds a concrete skeleton plan and maximizes the final reward of a sequence with a horizon $H_i = 1 + depth_i$. In combination with the UCB criterion for trees, it searches element-wise from it previous state to an optimal decision sequence. This limits the number of visits M of every tree node and V refers the information of this node. By limiting the number of visits, new created nodes will automatically explore the environment in the decision space.

Consider the Bellman equation in (3), each decision state value estimates a probability $p(z' \mid z, u)$, where z is a decision node and taking a decision u at a state s in a tree path π_s

$$V^*(w) = \begin{cases} r & \text{if } w \text{ is terminated} \\ \int_{z'} V^*(z')\mathrm{d}p(z' \mid z, u) & \text{else.} \end{cases} \quad (3)$$

After the expanding step, the nodes have to be updating using backpropagation for the information value $V^*(z)$.

$$V^*(z) = \begin{cases} r & \text{if } z \text{ is terminated} \\ \sup_u V^*(z, u) & \text{else.} \end{cases} \quad (4)$$

Algorithm 1: EXPAND-TEST

Input: $node$
if *node is a decision node* **then**
  s = streamFromNode(node)
  if *s.output is continuous* **then**
    return PW-LAW(node)
  else
    if *s.outputSpace* $\subseteq \{$*z.decisionHistory*$\}$ **then**
      return False
    else
      return True
    end if
  end if
else
  return True
end if

Algorithm 2: CHILD-SELECT

Input: $node$ **if** *node is a decision node* **then**
  $child\_node$ = UCB-SELECT($node$)
else
  $child\_node$ = LEAST-SELECT($node$)
end if
return $child\_node$

Using the Algorithm from Tianyu *et al.*, for expanding graph structure Alg. 1, selecting new child-node Alg. 2, and extending the tree search Alg. 4. Alg. 1 is responsible for the expansion of the individual state nodes. A new edge will be added to the graph structure when EXPAND-TEST($node$) returns the value $True$. Therefore, a new random child have to be selected. This new child node, refers in Alg. 3, and can be a new decision-node or a transition-node.

Taking a NEW-DECISION($node$) and adding a new child to the tree, the equation 1 uses the UCB formulation for selecting children nodes to the graph and take the maximum score value for the decision node. Otherwise, when a transition node is selected, it uses LEAST-SELECT($node$). Here, the least-visited child node will be into account. These two strategies are shown in Alg. 2.

Alg. 4 return a single concrete skeleton plan π_c. Before the algorithm is terminated, RECEIVE-VISIT($node$) increase the visit of a node by 1. It is necessary to explore the decision space, when EXPAND-TEST($node$) is $True$; it will randomly select an unvisited decision state for the skeleton branch. Meanwhile, for the transition nodes, it evaluates a random selected node in the skeleton plan and establishes a relation of a new child node.

After the expansion of the state node in the skeleton plan, the environment updates by backpropagation at each state node in the branch. By converging the reward of the skeleton branch by selecting and expanding nodes, the MCTS algorithm guarantee an optimal plan π_c

Algorithm 3: NEW-CHILD

Input: $node$
if *node is a decision node* **then**
  new_node = NEW-DECISION(node)
else
  new_node = NEW-TRANSITION(node)
end if
node.addChild(new node)
return new_node

Algorithm 4: EXTENDED-TREE-SEARCH

Input: P_s, t_{ts}
$extended\_root$ = buildExtendedRoot(P_s)
$node \leftarrow extended_root$
while *timeCost()* $< t_{ts}$ **do**
  while *node is not terminated* **do**
    RECEIVE_VISIT($node$)
    if *EXPAND-TEST*($node$) **then**
      $node \leftarrow$ NEW-CHILD($node$), see Alg.3
    else
      $node \leftarrow$ CHILD-SELECT($node$) see Alg.2
    end if
    backupRewardFrom($node$)
  end while
  $node \leftarrow extended\_root$
end while
π_c = highestValueBranchFrom($extended\_root$)

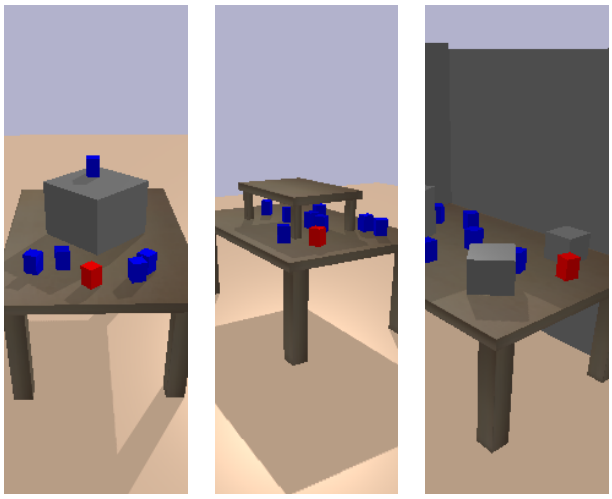
V. SYSTEM OVERVIEW

In the following section we evaluate the developed environment for the individual robot manipulation tasks; search for an executable plan, moving to the target and grasping the object without any collision. The policy should return a full feasible skeleton plan that describe every configuration in the environment. Moreover, an overview about the development environments is shown in Fig. 3 for the individual robot manipulation tasks to execute and compare various specialized environments.

A. Motion Domain

Robot manipulation task requires an intrinsic offline process that relies on domain description [3]. Sometimes, priorities must also be set during the searching phase. For these cases, different specifications must be made in the simulation domain. In this way, the robot needs to be able to perform on different aspects in a real world environment behaviors.

To represent these simulated environment we use the simulation PyBullet [31]. To support a heuristic search in the TAMP, it requires efficient updates in the scene. Nonetheless, the role of a planner usually consists of different goal determinations. TAMP is used here as getting an initial configuration of the domain and its obstacles and searching



(a) Full reachable targets (b) Blocked grasping directions (c) Unreachable targets

Fig. 3. Example Environments: Different aspects should be considered in the crafted world domains. The environment of Fig. 3(a) should verify the reliability of the algorithm. For more complex search domain, represent Fig. 3(b) and Fig. 3(c) combines difficult domain space and unreachable target objects

a goal topology of the end specification. The MCTS tree search should give us a feasible state at each skeleton node with the PDDLStream formulation. The execution between individual states, for example, the start and end conditions will not further consider.

B. Skeleton Plan

Long-term hierarchical structures can be exceptionally expensive. Therefore, a certain limitation is required to optimize states in skeleton plans. The geometric properties, for example, can be used to get a more restrict grasp direction. The objective TAMP of preparing the different behaviors in a real task is the concrete requirement description. Therefore, the description of a feasible plan cannot always be fully described with symbolic constraints, as the complexity of a task increases. The Table V-B shows a possible plan to be executed with symbolic placeholders for reaching a target object and grasping an object given in Fig. 4 environment. Here, the output in a member operator can be described as input for the next subtask.

TABLE II

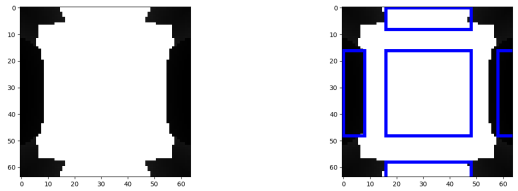
FEASIBLE SKELETON PLAN FOR SIMPLY GRASPING A TARGET BODY USING PDDLSTREAMS.

Behavior	Member operators
Approach to pick	Sample-grasp-direction$(\cdot) \rightarrow \#$ Sample-grasp$(\cdot, \#) \rightarrow \#$ Sample-base-position$(\cdot, \#, \#) \rightarrow \#$ Plan-free-motion$(\cdot, \#) \rightarrow \#$ Move-free-base$(\cdot, \#, \#)$ Inverse-kinematics$(\cdot, \cdot, \#) \rightarrow \#$ move-free-arm$(\cdot, \#)$ Pick$(\cdot, \cdot, \#, \#, \#)$

1) Sample-grasp-direction: Each object s_0 in a environment space provides an optimistic number of grasping directions $d_i \subseteq D$. This includes, for example, all geometric grasp directions proceeding from the movable object s_0.

$$D = \{+Z, +X, -X, +Y, -Y\} \tag{5}$$

The decision of a successful grasp in motion planning can be considered under several conditions. The simplest decision approach is based on a naive assumption, where one of the grasp direction d_i is randomly selected. This method will randomly select one of the grasping directions w.r.t. to the target object. There is no attention to whether the object can also be moved without any collision, or the path of the object side is available by the randomized selection. Appropriate methods and metrics include the direct environment of the target object.



(a) (b)

Fig. 4. Bottom-up environment perspective of an target body, where the white color represent the a graspable directions and the black color represent a blocked direction. Here, Fig. 4(a) shows the direction of $+$Z, $-$Y and $+$Y are feasible and the direction $-$X and X are blocked. This makes it possible to include the immediate environment for the object and this is shown in Fig. 4(b)

The heuristic approach uses the immediate environment as an initial value, and this is determined by a camera or sensors. Fig. 4 visualize the perspective of a target object in an environment. This makes it possible to draw better decision boundaries and only selects possible grasping direction by a trained CNN. This has the benefit that the surroundings in the environment are more considered. Eventually, only possible combinations are examined in the MCTS algorithm. This heuristic approach reduces computation cost and planning time.

2) sample-base-position: One of the most important tasks is the positioning of the robot manipulator. This task can be really hard finding a suitable position, because of different requirements of the strategy. There is multiple ways for describing a plan of actions, whether to take a greedy approach for getting the first suitable position, that lies in the working space of the robot to grasping a target object or a long-horizon position for getting the shortest position in relation to the start position and taking different aspect of the environment into account.

The Fig. 4 shows an open environment, where the target object on the table can be grasped on multiple sides. Sometimes it's not very suitable, if the robot has to go around the table to a goal position for grasping the target object. Therefore, the trade-off of finding a goal position as fast as

possible like in a greedy way and or a long-horizon approach for find the best position, but this needs more computation time.

The position of the robot is determined randomly. A threshold, which represents the maximum distance to the object, as a possible workspace of the robot can be determined. In addition, the confidence of the randomized position can be evaluated here by an NN, which determines the current position by assuming the grasps direction and sample grasp.

The heuristic pose generator uses the same algorithm, except for evaluating an acceptable position. A NN takes the decision, if the grasp will be successful from the found position in the future MCTS tree search. This has the advantage, that, if the robot wants to grasp an object from a certain direction, the trained NN decides if the current position is reliable.

3) Inverse-kinematics: Based on the grasping direction and the found position, the inverse kinematics is calculated with the intern PyBullet-API [31]. This method is intended to determine the end effector of a robot manipulator to find a configuration of all joints in order to grasp the target object without any collision.

VI. EMPIRICAL EVALUATION

In this section, we will empirically evaluate the MCTS algorithm with UCB for exploring tree policies. A robot should grasp different objects on a table with different levels of difficulties: open environments, multiple static objects in different sizes, and environments that blocks multiple grasp directions. Some representation of these tested environments are shown in Fig. 3

The planner has not received a termination criterion for solving each of the tested environment. Nevertheless, for the search of a position and the inverse-kinematic, our approach has a maximum iteration limit for finding a solution. The evaluation of the MCTS algorithm takes about 400 tests of each scenario. When a feasible plan is found, the simulation stops, and it will calculate computation time. This time also includes calling the skeleton plan and loading the scenarios into the simulation environment. Each environment scenario initializing arbitrary geometric boxes randomly on a placed region and loaded from a configuration file. This allows comparison between the randomized decision and heuristic based decision approach at each scenario. In order to be able to make a statistical statement about the quality of the MCTS algorithm, each target object in the environment was approached five times. As a result, almost 1600 tests were executed on both configuration.

A. Accessible objects

The motivation of this task, shown in Fig. 3(a), is to evaluate the reliability of the MCTS algorithm, where every object in this scenario is reachable. The tendency of an easy environment scenario using a CNN to pre-select grasp direction leads to better results than the randomized decision. The consequence is, that not all possible combinations needs to be tested in the environment space. Supporting different neural networks strategies for searching the right decision and finding a concrete skeleton plan, the computation time can be reduced by **72.22** %.

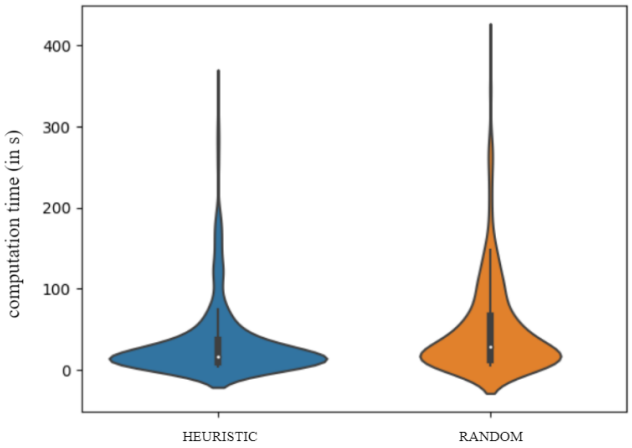


Fig. 5. Evaluation: MCTS tree search evaluation on different modes. First capturing scenarios on heuristic approaches and second on a random based decision approach. The approach of the tested domains in is shown in x-axis and on the y-axis are the samples of computation time for finding a successful skeleton plan

On average, the random decision search can solve a simple scenario in **27.7** s, meanwhile the heuristic search needs **7.5** s. In both, the randomized search and heuristic search becomes sometimes more costly, because the state nodes in the skeleton plan are unable to find a suitable inverse kinematic configuration for the robot without any collisions or violating the joint space. In combination with the MCTS tree search and supporting the search algorithm with a heuristic search for finding an executable skeleton plan, this can be increased by approximately **6** %.

B. Analysis of the results

The Fig. 5 shows the computation time at each policy method, which includes all the test domains. The y-axis shows the computation time (in s) during the algorithm's policy search at all possible scenario configuration, and the x-axis shows the number of all possible execution time outcomes at each test.

In the section VI-A, we already indicate that the computation time in heuristic search is giving us faster and confident solutions. These results solidify in all other environment scenarios. The Fig. 2 shows an example of a policy tree based on MCTS tree search. This reinforcement algorithm uses the knowledge on the environment to take decision by actions. The Fig. 5 also illustrates, that in all scenario problems, it is more frequent and likely that the heuristic search finds a faster solution. Over all scenarios, the heuristic search only needed half of the computation time to find a solution. A major problem, for the large distortion in the search algorithm, is to find an inverse kinematics configuration based on the PyBullet-API. Our approach mostly fails because the algorithm run out of iteration or finding an appropriate inverse kinematic solution.

VII. CONCLUSION

There is a massive interest in task and motion planning algorithms. We propose in this paper a general approach for hierarchical planning with a long horizon strategy that uses symbolic constraints. MCTS integrates a tree search that binds a concrete skeleton plan with an action space. UCB allows converging to through the tree search for finding an optimal plan and take decisions to explore the environment space. Based on the exportation-exploration, it has the penitential for speeding up existing planning algorithms for selecting hierarchical operators and will be able to improve best decisions over time. The empirical result also reveals us, that a naive approach for a MCTS tree search takes longer computation time instead of an AI supporting approaches that takes previous decision to the MCTS algorithm for supporting the success of the decision. In order to apply uncertainty computation time outcomes, and failing particularly observable environments, it will be better in future works to briefly improve the inverse kinematics to ensure the efficiency in long horizon decision planner. This ensures the capabilities for comparison to other planning generator methods.

REFERENCES

[1] B. Kim, Z. Wang, L. Kaelbling, and T. Lozano-Pérez, "Learning to guide task and motion planning using score-space representation," *The International Journal of Robotics Research*, vol. 38, p. 027836491984883, 05 2019.

[2] T. Lozano-Pérez, "Spatial planning: A configuration space approach," *Computers, IEEE Transactions on*, vol. C-32, pp. 108–120, 03 1983.

[3] T. Ren, G. Chalvatzaki, and J. Peters, "Extended task and motion planning of long-horizon robot manipulation," *CoRR*, vol. abs/2103.05456, 2021. [Online]. Available: https://arxiv.org/abs/2103.05456

[4] 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 *IEEE International Conference on Robotics and Automation (ICRA)*, 2014. [Online]. Available: http://robotics.eecs.berkeley.edu/pubs/25.html

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

[6] C. Garrett, T. Lozano-Pérez, and L. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," 10 2020.

[7] D. Long, H. Kautz, B. Selman, B. Bonet, H. Geffner, J. Koehler, M. Brenner, J. Hoffmann, F. Rittinger, C. R. Anderson, D. S. Weld, D. E. Smith, M. Fox, and D. Long, "The aips-98 planning competition," *AI Magazine*, vol. 21, no. 2, p. 13, Jun. 2000. [Online]. Available: https://ojs.aaai.org/index.php/aimagazine/article/view/1505

[8] N. Shah and S. Srivastava, "Anytime integrated task and motion policies for stochastic environments," *CoRR*, vol. abs/1904.13006, 2019. [Online]. Available: http://arxiv.org/abs/1904.13006

[9] P. Regier, A. Milioto, P. Karkowski, C. Stachniss, and M. Bennewitz, "Classifying obstacles and exploiting knowledge about classes for efficient humanoid navigation," in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, 2018, pp. 820–826.

[10] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018. [Online]. Available: https://doi.org/10.1177/0278364918761570

[11] N. Dantam, Z. Kingston, S. Chaudhuri, and L. Kavraki, "Incremental task and motion planning: A constraint-based approach," 06 2016.

[12] J. Sierra-Santibáñez, "Heuristic planning: A declarative approach based on strategies for action selection," *Artificial Intelligence*, vol. 153, no. 1, pp. 307–337, 2004, logical Formalizations and Commonsense Reasoning. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370203001504

[13] C. Weber and D. Bryce, "Planning and acting in incomplete domains," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 21, no. 1, pp. 274–281, Mar. 2011. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/13463

[14] M. C. Fu, "Monte carlo tree search: A tutorial," in *2018 Winter Simulation Conference (WSC)*, 2018, pp. 222–236.

[15] Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Transactions on Computers*, vol. C-32, no. 2, pp. 108–120, 1983.

[16] E. Xidias, P. Azariadis, and N. Aspragathos, "Path planning of holonomic and non-holonomic robots using bump-surfaces," *Computer-Aided Design and Applications*, vol. 5, pp. 497–507, 08 2013.

[17] M. Fox and D. Long, "Pddl2.1: An extension to pddl for expressing temporal planning domains," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 61–124, 12 2003.

[18] B. Kim, L. P. Kaelbling, and T. Lozano-Pérez, "Learning to guide task and motion planning using score-space representation," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 2810–2817.

[19] J. Hoffmann and B. Nebel, "The ff planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research - JAIR*, vol. 14, 06 2011.

[20] S. James, G. Konidaris, and B. Rosman, "An analysis of monte carlo tree search," 02 2017.

[21] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, "Monte carlo tree search: A review on recent modifications and applications," 03 2021.

[22] T. Keller and P. Eyerich, "Prost: Probabilistic planning based on uct," 2012. [Online]. Available: https://www.aaai.org/ocs/index.php/ICAPS/ICAPS12/paper/view/4715

[23] P. Coquelin and R. Munos, "Bandit algorithms for tree search," 04 2007.

[24] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, pp. 235–256, 05 2002.

[25] A. Cotarelo, V. García-Díaz, E. R. Núñez-Valdez, C. González García, A. Gómez, and J. Chun-Wei Lin, "Improving monte carlo tree search with artificial neural networks without heuristics," *Applied Sciences*, vol. 11, no. 5, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/5/2056

[26] L. Kocsis, C. Szepesvári, and J. Willemson, "Improved monte-carlo search."

[27] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016.

[28] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[29] D. Speck, R. Mattmüller, and B. Nebel, "Symbolic top-k planning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 9967–9974, 04 2020.

[30] M. Katz, S. Sohrabi, O. Udrea, and D. Winterer, "A novel iterative approach to top-k planning," 2018. [Online]. Available: https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17749

[31] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.