Table of Contents

1 Introduction 2
2 Related Work 3
3 Formalisation 5
3.1 A Formalism for TAMP 5
3.2 Application to Tool Use 6
4 Problem Decomposition 7
4.1 Blackbox Concept 7
4.2 Optimality 8
5 Tool Use Example 10
5.1 High-Level Planning 10
5.2 Motion Planning and Interface 11
6 Evaluation 12
7 Discussion 17

An AI formalization of Betty the Crow's sequential geometric tool use

Lukas Sauer

Advisor: Prof. Dr. rer. nat. Marc Toussaint

Abstract. Betty The Crow was a new caledonian crow that, in the lab of Alex Kacelnik, demonstrated surprising skill in sequentially using tools to reach for other tools to reach for a reward. The goal of this work is to find an AI formalization of such behaviour that combines reasoning over a sequence of first-order logic decision variables as well as over the geometric path to execute the reaching and tool use motions. For that, we first consider the general area of Task And Motion Planning problems; we use an approach of decomposing the problem into smaller ones solvable by (existing, blackbox) modules; we test a simple implementation of the resulting method on some small problem instances.

Zusammenfassung. Betty die Krähe war eine Neukaledonienkrähe, die in Versuchen Alex Kacelniks erstaunliche Fähigkeiten zeigte, sequenziell Werkzeuge zu benutzen, um an andere Werkzeuge heranzukommen, um eine Belohnung zu erreichen. Das Ziel dieser Arbeit ist es, eine KI Formalisierung solchen Verhaltens zu finden, die Überlegungen über eine Sequenz von prädikatenlogischen Entscheidungsvariablen sowie über den geometrischen Pfad um die Greif- und Werkzeugnutzungsbewegungen auszuführen kombiniert. Dazu betrachten wir zunächst das allgemeine Gebiet von Task And Motion Planning Problemen; wir benutzen einen Ansatz, das Problem in kleinere, von (existierenden, Blackbox) Modulen lösbare, zu zerlegen; und wir testen eine einfache Implementierung des resultierenden Verfahrens an einigen kleinen Probleminstanzen.

1 Introduction

The ability to solve complex manipulation tasks is something very closely connected to the intuitive understanding of intelligence. It is present and has been highly important for humans, but can also be observed to an extent in some animals, for example in crows. At the same time, problems involving the manipulation of objects in a geometric environment are still not trivial to solve automatically. There appears to be a gap between being able to solve abstract problems and working in continuous geometric spaces; Because, at the moment, exhaustively exploring the latter and for example directly doing motion planning at a fine enough discretization is not feasible, current methods employ abstract planning to generate smaller subproblems, pruning the intractably large spaces. The class often denoted as task and motion planning problems (TAMP) has been an area of research for a long time.

This work follows that same thought, inspired by the behaviour observed in some crow specimens to use tools like sticks for object manipulation, and especially by their ability to use multiple, consequently larger tools to reach food in complex processes.

For example, a bird may use one readily available yet short hook to gain access to a second, longer one, and may use that to obtain a third by which it can finally fetch a piece of meat. The eponymous Betty the Crow was just one of these. Such indirect manipulation (touching and moving a tool via another one) can be challenging e.g. in that it vastly increases the number of grasps and relative positions to be considered; The complexity of the abstract domain is somewhat increased, too. Our attempt is a strict separation of high-level and motion planning. We will use existing modules for both parts of the problem and a very simple interface layer in-between to integrate them.

Our aim is to provide a theoretical formulation of the problem, to show that an implementation in separate components can in principle find optimal solutions to such problems, and to provide a first toy implementation.

The structure will be as follows: In the next section, we cover some of the extensive existing literature on TAMP or manipulation planning; After that, we present a formulation of the problem per se and lose some words as to how sequential tool use can be represented in it. In Section 4, we consider a strict decomposition of such problems according to the TAMP term as well as its potential to be solved optimally; In Section 5 follows a description of the tool use problem in that frame. Section 6 offers some experimental results, and Section 7 a brief final discussion.

2 Related Work

TAMP is an area well known to be interesting and as such, there is a large number of works in the field. We briefly mention some of them, mostly more or less recent, which were reviewed and are relevant to the subject.

Work in this general area has been going on for quite some time; An early work is [LPJM+89], which describes the task-level system Handey that operates on highly abstract tasks, effectively being a lower-level half of a Task And Motion Planner.

What we perceive as the classical decomposition for complex manipulation problems is that which leads to the TAMP term: One does high-level planning and tries to do corresponding motion planning on the result. More recent contributions to this field are for example [LDSK12], where a problem called geometric backtracking is identified - the (infinitely, or even discretized still vastly) many geometric instantiations of a symbolic state or action. They make use of (linear) constraints and an intermediate layer managing the shrinking intervals of possible values for variables before doing the actual backtracking, thus eliminating many possible attempts beforehand; Their following paper [LDB+14] goes into more detail. [LPK14] generate so-called plan skeletons, which contain constraint variables, and use discrete constraint satisfaction problem solvers on these. In [Tou15] symbolic search is done over action sequences and subsequently the result and intermediate poses (and, eventually, trajectories) are optimized. The problem posed is not exactly a goal state, but an evaluation function over end states.

A direction pursued in some works is to enrich a symbolic planner with predicates that depend on the actual geometric state, often referenced as semantic attachments. [DEK+12] covers this, but they deliberately avoid giving the underlying geometric calculations any choice; Thus, their approach suffers from drawbacks in complex manipulation settings where the intermediate placement of an obstacle may be of consequence later. [EHP+11] also use a task planner with conditions on the geometric state that is explored and externally computed predicates. They consider scenarios with multiple robots and update the task-planning problem according to geometric results; In their examples, they consider a coarsely discretized 2D environment. The approach in [GLPK15] is similar in that they also do classical high-level planning with semantic attachments computed on-demand; They maintain graphs for the detailed geometric representation, using roadmaps and heuristics calculated from a relaxed plan graph. [SFR+14] is another work following the classical decomposition; But they explicitly employ an interface layer between the task planner and the motion planner. On this, symbolic references (to all possible grasp poses, possible trajectories etc.) are instantiated and iterated over. The interface layer can also return new symbolic information like the fact of one object obstructing the path to another, making for a tighter connection. The more recent paper [CHMS+] follows the same vein, but uses reinforcement learning to efficiently sample configurations for the generated high-level plan.

A slightly different take on complex manipulation problems is to think about the configuration space and subspaces of it where transit and transfer possibilities (moving the robot around and moving an object grasped by the robot, respectively) intersect. This is followed in [SLCS04], where these subspaces are managed in a graph (and sampled in roadmaps); Or also in [LP15], where a so-called grasp-placement-table is maintained, from which once again a graph is built.

[CAG09] offers an elaborate work on TAMP in general, as well as an approach where roadmaps are computed for certain propositions and a path is searched in the combination of these roadmaps. They use the symbolic consequences of actions as a heuristic to guide the exploration. [SPGA13] uses hierarchical task networks, but the principle of iterating over candidate grasps, placement positions etc. is basically the same.

All these papers cover some aspect of TAMP; To our knowledge, none of them tackle the problem of sequential tool use considered here. The theoretical formulations vary quite a lot, sometimes not being explicitly given.

In later sections of this paper, we use existing technologies for a decomposition of the problem. The high-level aspect uses STRIPS rules, going back to [FN72], albeit heavily modified. The motion planner uses standard optimization methods; For more detail on this subject, we refer to [BV04] and [NW06].

The behaviour of (meta-)tool use witnessed in crows is covered, for example, in [THHG07] (extraction of one tool using another), [WWC+09] (sequential tool use and choice between several candidate tools), where the reader can also find more information on Betty, and [WWK11] (crows using tools for non-foraging activities, like interacting with potentially dangerous objects). [VMAK10] offers an interesting perspective on the mechanisms involved in the choice process (of starlings, in this case).

3 Formalisation

3.1 A Formalism for TAMP

In the following, we define a TAMP problem: In a general motion planning problem we get

- an n-dimensional world configuration space X (i.e. also containing the position and orientation of all dynamic objects)
- the domain of paths in X $P = \{(c, T) | T \in \mathbb{R} \wedge c : [0, T] \to X\}$
- a feasibility function $f : P \times \mathbb{R}^+ \times \mathbb{R}^+ \to \{0, 1\}$, which is true for path $p = (c, T)$ iff it is consistent - no collisions occur at any time, objects behave as they are supposed to with regard to pushing and pulling etc. - in the time interval between the times given as first and second real parameter
- an initial state $x_0 \in X$
- a goal subset $E \subset X$ of world end configurations

and try to find a feasible path (one for which f evaluates to true over 0 to T) that starts in x_0 and the end state of which is in the goal set. Further, we may have

- a cost function $e : P \times \mathbb{R}^+ \times \mathbb{R}^+ \to \{0, 1\}$, which gives the corresponding costs in the given time frame

that we try to minimize:

$$\begin{aligned} \underset{p=(c,T)}{\text{minimize}} \quad & e(c, 0, T) \\ \text{subject to} \quad & c(0) = x_0, \\ & c(T) \in E, \\ & f(c, 0, T) = 1 \end{aligned}$$

A TAMP problem additionally requires

- a logic L, in our case, a tuple $< S, O >$ of a set of predicate symbols S and a set of objects O; By this we effectively employ database semantics (closed-world-assumption and unique names)
- an initial state $l_0 \in L$ which basically describes how the predicates evaluate on the objects
- a set $A : L \to L$ of action operators that map a logical state to its successor
- a goal set $G \subset L$ (usually defined via requirements on facts to hold or not hold)
- and finally a set Σ of extensions of the predicates $s \in S$ with $\sigma : dom(s) \times X \times \{0, 1\} \to \{0, 1\}$ that evaluates to true if on a symbolic level, s evaluates to the given truth value with the given arguments in configuration x

Our solution now also contains a sequence $(a_k, l_k)_{k \in 1,..,K}$ of actions and successor states as well as a set of k ordered times $\tau = \{t_k \in \mathbb{R}^+ | k \in 1, .., K \wedge t_{k-1} < t_k\}$ $(t_0 = 0)$. To sum up, we search for a solution (path $p = (c, T)$, sequence $(a_k, l_k)_{k \in 1,..,K}$ and timings τ) for which

- $\forall_{k=1:K} : l_k = a_k(l_{k-1})$ - for all k l_k is the result of executing a_k in l_{k-1}
- $l_K \in G$
- a fact $s(o)$ (predicate s on object vector o) holding in l_k implies that $\sigma(o, c(t_k), 1)$ holds, not holding implies that $\sigma(o, c(t_k), 0)$ holds (represents consistency of the symbolic and geometric representation)

Additionally, the path must still solve the geometric problem

- $\forall_{k=1:K} : f(p, t_{k-1}, t_k) = 1$
- $c(0) = x_0$
- $c(T) \in E$

and (potentially) minimize over e. Written more mathematically again:

$$
\begin{aligned}
\underset{p=(c,T),K\in\mathbb{N}^+,(a_k,l_k)_{k=1:K},\tau}{\text{minimize}} \quad & e(c, 0, T) \\
\text{subject to} \quad & \forall_{k=1:K} : l_k = a_k(l_{k-1}), \\
& l_K \in G, \\
& \forall_{i=1:|S|}\forall_{k=1:K}\forall_{o\in dom(s_i)} : s(o) \in l_k \Rightarrow \sigma(o, c(t_k), 1) = 1 \\
& \forall_{i=1:|S|}\forall_{k=1:K}\forall_{o\in dom(s_i)} : s(o) \notin l_k \Rightarrow \sigma(o, c(t_k), 0) = 1 \\
& \forall_{k=1:K} : f(p, t_{k-1}, t_k) = 1 \\
& c(0) = x_0, \\
& c(T) \in E,
\end{aligned}
$$

In this formulation, the geometric aspect is stronger than the symbolic one and merely augmented by it; indeed we can represent arbitrarily little of the geometric information in it and could even degenerate it to a partial problem that is already and always solved.

3.2 Application to Tool Use

Let us consider what that formulation would look like for tool use as observed in some crow specimens. The motion planning problem's configuration space is exactly that, the world configuration space - the positions and rotations of all movable elements, as well as position and configuration of our actor, crow or robot. The feasibility function is hard to formulate in anything other than a very abstract way - it contains freedom from collisions, the fact that a tool can be pulled with another tool and will move if pushed or pulled, even things like the effects of gravity on objects not held or only held in a not altogether rigid way. This is not only difficult to state, but even more so to actually implement. In this paper, the world behaves in highly simplified ways: gravity is not in effect, and once tools have touched, they can stay connected in a rigid way arbitrarily long and exactly until not required any more. The initial state and goal subset are straightforward, even though the latter is, again, easiest to give in abstract as all states where the desired object is directly connected to our actor.

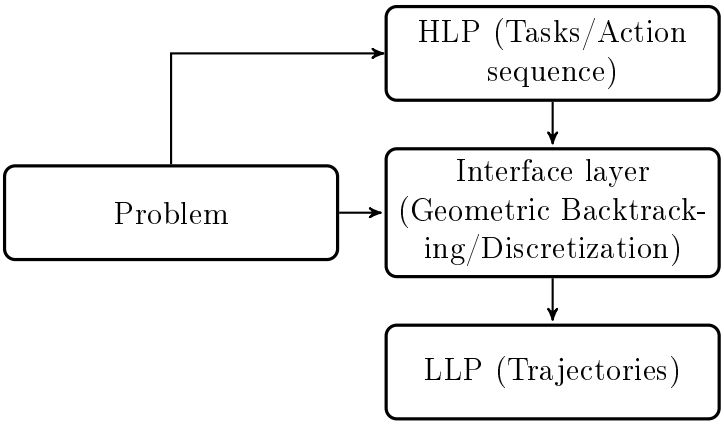
The task part of the problem has already been mentioned to be only a help in solving it. We could formulate it arbitrarily weak, but that would gain us nothing in computational savings. What we settle for is a logic describing which tools or objects are connected to which, action operators modifying exactly these predicates, and the goal set just as formulated in the motion planning problem. The extensions of the predicates are problematic in that whether objects are connected in a situation may depend on whether we want them to be - the implementation that comes to mind is one that denies objects not touching each other being connected or confirms them not being connected, whichever has to be checked, and confirms whichever query is made for objects that are in contact.

The intuitive cost function over trajectories is one that penalizes movement, sudden changes of direction (first derivative of positions), and the passing of time.

4 Problem Decomposition

4.1 Blackbox Concept

The intuitive decomposition of a complex motion planning problem is the one that leads to the TAMP concept - namely, to search for an abstract high-level plan that highly restricts the motion planning space in each step. We consider the components of this decomposition further:



HLP: The high-level planning layer hands down an abstract representation of the tasks at hand. This mirrors what we saw in the problem definition: We transfer part of the computation to a classical planning problem; Exactly how much corresponds to how detailed/informed the actions in this formulation are. Note that we do not necessarily hand down only complete plans - for a cost and/or feasibility evaluation, we may also query just the first few steps of a task plan.

LLP: The low-level layer is a very straightforward motion planner. It gets initial and end configurations - poses as well as the configured environment. It does trajectory planning and returns feasibility information or costs of the optimal trajectory (In other approaches that aim for more communication between the layers, it would also pass on indicators as to what caused the trajectory planning to fail).

Interface layer: This layer gets the abstract plan or partial plan, and hands down instances of trajectory planning. It has to define the method for searching the reduced configuration space of the symbolic plan; This leads to an instantiation of configurations at the intersections of actions, so we know for each what configuration held before and which holds after. Between these, trajectory planning is executed. We then get the returned information, and eventually pass it on to the abstract layer. Again, this is feasibility information or costs (or an indicator of what caused the low-level planner to fail).

4.2 Optimality

An important consideration would be the optimality of this approach: Are we guaranteed to reach solutions if such exist, and optimal solutions if we have a cost function? Or, following a different train of thought, what assumptions do we need to make to reach such guarantees? We will consider this step-by-step, or layer-by-layer.

For the highest level, we do classical planning. This is a rather well-known problem, and there exists a variety of approaches and algorithms for it. For optimality, it helps to think about graph search over an infinite state graph. There, optimality depends on our cost function - do we have edges with zero-weights? This question is somewhat philosophical. The intuitive approach to cost functions is to penalize movements, because after all, that takes actual effort. But in our tool use example, some symbolical actions may not require such. Think of 'connecting' a huge series of toy cubes to remove them from the path to an obstacle. We would have to do an enormous number of connects; but the actual movement required would be as small as to clear a direct way. The extreme of this might be a situation where the cubes already touch each other. In the formalism, we would still require all those connective actions that would indeed have zero costs, and a graph search algorithm might deem this not worthwhile and try something else. Optimality appears rather hard to reach in that case. Slightly less pessimistic or restricting the problems we work on, we could plan optimally if we had a finite branching factor (a finite set of actions to execute in each state, which is a sensible assumption to make) and non-zero costs in all cases. What we also require is an evaluation of the costs of an action, or a helpful heuristic.

The problem with this is that optimality of a substep may change considering a longer plan. The placement of an obstacle in a substep is done optimally by placing it at little cost first - but if that restricts later movement, a different, more expensive positioning gives a better overall plan. Evaluating the cost of a single step is not trivial. We can, however, try to optimize over the entire plan up to the current point in time; This whole phase is then, by definition, optimally cheap and can only get more costly if it is part of a bigger optimization again. Thus, we can view the computed costs as an optimistic estimate of the actual costs. In other words: We evaluate the costs of a state, which is slightly different than calculating those of a single step or transition between states.

If this estimate is too optimistic, it doesn't really help us. But using these optimal costs should be a good start. Thinking about cost functions, it might also make sense

to penalize the passing of time or the cognitive/computational load associated with each step. This makes for non-zero minimal costs - in fact, we could even set a fixed lower bound for each step. If that is the case, we can put a bound of $O(b^{c^*/\epsilon})$ on the number of states we need to evaluate for a maximum branching factor (i.e. number of possible actions in a state) b, optimal costs c^* and a lower bound ϵ on the step costs. This may be an intimidating figure, depending somewhat on how we weigh the basic costs of a step. Also, each of these state evaluations corresponds to a full problem or task plan for the interface layer to solve optimally.

On a lower level, optimality is connected to our optimization algorithms. We get a start and end configuration with constraints on freedom from collision and such, and try to optimize the actual motion. This is not necessarily easy; With obstacles, we get discontinuous costs or constraints, and e.g. a more sophisticated world model for tool use would need to consider dynamic connections of objects and more. It boils down to what assumptions we can make about the real cost function over trajectories. If it is arbitrarily malicious, we are doomed - we cannot hope to exhaustively search a real-valued configuration or control space. On the other hand, it is not too unrealistic to assume a smooth cost function (sum of squared manipulations and derivatives, or movements in space) with constraints to incorporate the circumstances mentioned above. As such, optimality may be highly computationally expensive but is to our understanding technically feasible.

The mid/interface level would need to determine the start and end configurations for trajectories. This is a huge space to search - e.g. finding all configurations such that the agent is indirectly connected to some object includes all possible connections, all agent positions, etc. - but it is definitely an improvement over just optimizing the entire trajectory at once, with no guidance from a high-level plan, which is the entire point of the TAMP decomposition. Thinking about optimality, we can again view this as a blackbox optimization problem. This one may be slightly more malicious, since the costs can be expected to show some discontinuities at extreme placements of obstacles, but again, it should be theoretically feasible.

To sum up, we need to make those assumptions to make the two optimization stages optimally solvable. If just the trajectory planner is non-optimal, we already get non-optimal overall solutions. We need optimality in the interface layer to get a useful state cost evaluation for the upper layer; with that, high-level optimality is reachable.

5 Tool Use Example

5.1 High-Level Planning

We consider tool use/indirect manipulation as an example of TAMP, e.g. as has been demonstrated by crows. The problem is to reach a desired item (a tool, in this formalization, but in practice usually food), that is generally not reachable directly. For this, we give the high-level problem in a variant of STRIPS rules:

Predicates/Conditions:

- $grabbed(X)$, designating X to be directly in the grasp of the robot or actor
- $connected(X, Y)$, which symbolizes X via Y (e.g. hooking into X with the hook Y)
- $grabbable(X)$
- $connectable(X)$
- $freeToBind(X)$, meaning X can be used to connect other tools
- $freeToBeBound(X)$, meaning X can be grabbed/connected
- $freeToBeUnbound(X)$, making a disconnect/letGo action on X meaningful
- $isTool(X)$
- $isBase(X)$

Operators:

$$grab(X, Y)$$
$$\{\{isTool(X), isBase(Y), freeToBeBound(X), grabbable(X), freeToBind(Y)\},$$
$$\{freeToBeUnbound(X), freeToBind(X), grabbed(X), freeToBeBound(X)!,$$
$$freeToBind(Y)!\}\}$$

$$letGo(X, Y)$$
$$\{\{isTool(X), isBase(Y), freeToBeUnbound(X), grabbed(X)\},$$
$$\{grabbed(X)!, freeToBeUnbound(X)!, freeToBeBound(X), freeToBind(Y),$$
$$freeToBind(X)!\}\}$$

$$connect(X, Y)$$
$$\{\{isTool(X), isTool(Y), freeToBeBound(X), freeToBind(Y), connectable(X)\},$$
$$\{freeToBeUnbound(X), freeToBeBound(X)!, connected(X, Y), freeToBind(X)\}\}$$

$$disconnect(X, Y)$$
$$\{\{isTool(X), isTool(Y), freeToBeUnbound(X), connected(X, Y)\},$$
$$\{freeToBeBound(X), freeToBeUnbound(Y), connected(X, Y)!, freeToBind(X)!\}\}$$

$$boundObstacleMakeConnectable(X, Y)$$
$$\{\{isTool(X), isTool(Y), freeToBeBound(X)!, connectable(Y)!\},$$
$$\{connectable(Y)\}\}$$

$$boundObstacleMakeGrabbable(X, Y)$$
$$\{\{isTool(X), isTool(Y), freeToBeBound(X)!,$$
$$freeToBeBound(Y), grabbable(Y)!\},$$
$$\{grabbable(Y)\}\}$$

$$boundToolMakeGrabbable(X)$$
$$\{\{isTool(X), freeToBeBound(X)!, grabbable(X)!\},$$
$$\{grabbable(X)\}\}$$

These operators represent intentions on the task level; whether or not they are feasible is subsequently checked on the interface layer. Each of them contains a set of preconditions and a set of effects.

Rules:

$$\{\{connected(X, Y)\},$$
$$\{freeToBeUnbound(Y)!\}\}$$

$$\{\{grabbable(X)\},$$
$$\{connectable(X)\}\}$$

Rules are automatisms of the world in this formulation; They are applied immediately if relevant.

Initial state:

- $isBase(Base)$
- $isTool(T)$ for all tools T
- $grabbable(T), connectable(T)$ according to an initial evaluation
- $freeToBind(Base)$
- $freeToBeBound(T)$ for all tools T

Goal state:

- $grabbed(Tool_n)$ for goal $Tool_n$

5.2 Motion Planning and Interface

Motion planning in our case is optimizing over the trajectory, an instance of constrained optimization. As mentioned before, the constraints incorporate requirements such as freedom from collision, contact for grasps and the likes.

The interface layer in principle optimizes over the configurations at the task intersections. Usually, this is done by sampling discrete intermediate poses; Other options and details on this are the subject of most of the papers referenced in Section 2.

6 Evaluation

Following the decomposition above, we tried to get a working miniature model of our tool use case. For this implementation, we simplified again a great deal: As mentioned before, effects like gravity were ignored and grasps were seen as sticky exactly as long as intended.

High-level planning took place with existing first-order logic planners, using basically Monte Carlo Tree Search. The cost function was set to be just the passing of time, making each step identically expensive on the upper level - costs evaluated on the lower levels were ignored for this purpose. The interface layer was done in simple python scripts with shell calls to the supplied modules; We used plain text files for communicating a blacklist of failed high-level plans to the task planner, extracting the actions from the task planner's output, generating the constraints and costs for each trajectory optimization step and evaluating the results calculated. Optimization here just took a heuristically helpful sample pose for intermediate positions in single instantiation trials and a small number (between 1 and 4) of such poses in geometric backtracking trials, thus making no attempt at actual trajectory optimization. Motion planning took place with the supplied KOMO module, which optimizes a trajectory from a given start point subject to given costs and constraints. With this, tools were excluded from collision detection; Reachability was regarded as a problem of collision between robot and table or robot and other deliberately placed obstacles. Computationally, it would be possible to save a significant amount of time if already optimized subpaths were cached somehow; We did not do anything in that direction.

In short, we did not attempt a full implementation of the decomposition pictured above, but only explored in a limited way the interfacing of the existing components. We did not aim for an optimal solution but simply for a procedure to find any feasible solution, with hopes of it being simple and/or inexpensive. All tests were done on a retail laptop, with an Intel Core i7 5700HQ CPU and 7.7GiB RAM.

Scenario 1	single instantiation	solved within 1432.5s
	geometric backtracking	solved within 1011.3s
Scenario 2	single instantiation	solved within 34.9s
	geometric backtracking	solved within 28.0s
Scenario 3	single instantiation	solved within 1264.5s
	geometric backtracking	failed, runtime >1h
Scenario 4	single instantiation	solved within 233.3s
	geometric backtracking	solved within 254.7s

Fig. 1. Results of the tests in single instantiation and geometric backtracking case (several candidate poses tried for motion planning).

As we see, the implementation works, in a limited way, on a limited selection of small examples. In Scenario 1, it is able to find a correct sequence of interactions: Use Tool 1 to connect to Tool 3 (the goal, here), move Tool 3 somewhere accessible, and grab it. The single instantiation version sometimes succeeded not with that task plan,

grab(T1,Base),connect(T3,T1),boundToolMakeGrabbable(T3),disconnect(T3,T1),letGo(T1,Base),grab(T3,Base),
but achieved the same result when instead trying to move Tool 1 to an accessible pose,
grab(T1,Base),connect(T3,T1),boundToolMakeGrabbable(T1),disconnect(T3,T1),letGo(T1,Base),grab(T3,Base),
This demonstrates the very loose correspondence of the boundToolMakeGrabbable step in the task plans and the actual trajectory produced. Here, actual optimization over possible target poses would be most necessary; Similar considerations apply to the letGo and disconnect actions. This is to some extent caused by the implementation not having a direct move action; but that would have blown up the explored plan space even more.

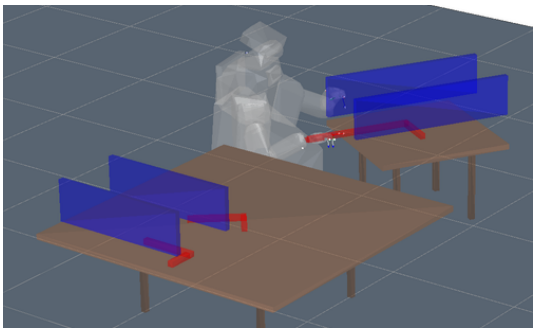


Fig. 2. Scenario 1: The small tool in front is the target; Reaching it requires the help of the long tool.

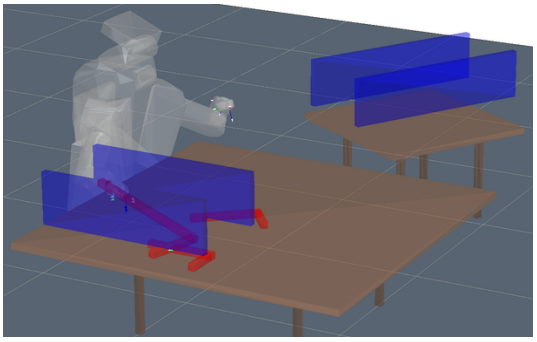


Fig. 3. With the long tool, the robot can reach the target.

The timing results with Scenario 2 are slightly unexpected - geometric backtracking always does at least the same work as single instantiation to find the same plan, but here it actually took less time for it. This leads to another problem with the implementation and its tests: runtimes varied strangely, the same optimization step sometimes being done almost instantaneously and sometimes taking tens of seconds. Small figures in the results are not really reliable; Over longer runs, the effects on the many subcalls mostly cancel out. Another observation with this setting is that the trajectory planner might be too strict. The target is in principle reachable with the first step; But the motion planner was unable to produce a sufficient turning motion.

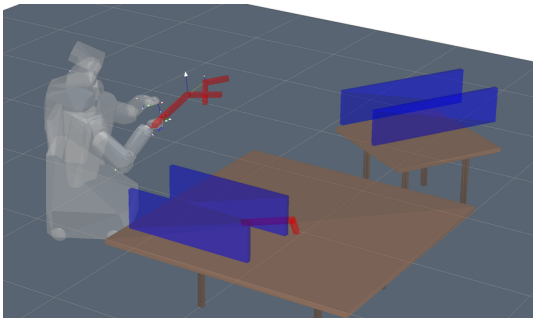


Fig. 4. To grab tools, they are moved into the free space. This was a heuristic for the letGo, disconnect and boundToolMakeGrabbable actions. Here, a step from Scenario 1 is shown.

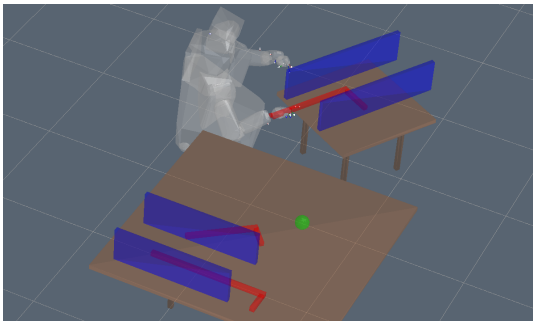


Fig. 5. Scenario 2: The target is the green ball, reachable via the long tool in front.

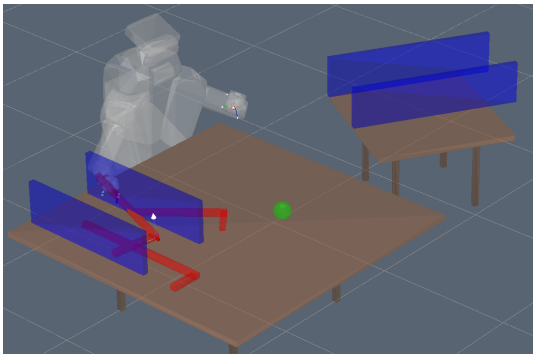


Fig. 6. The robot retrieves the required tool.

Scenario 3 demonstrates the very tight limits of the implementation: The setting contains another object and is solvable in some ten steps. The geometric backtracking version suffered from the exponential blowup with the number of steps - it was unable to find a solution after over an hour of testing, and we eventually cancelled the run. By that time, it had finished testing only some 15 task plans and was nowhere near a working solution. Our implementation spent nearly all of the time in motion planning; either more powerful machines, faster optimization or methods that avoid some of these computations seem necessary. The single instantiation run was, surprisingly, not much slower than on smaller settings; It also produced a plan that was very far from optimal with over 30 steps, but did work. It would seem that MCTS on the higher level tended to produce longer, more convoluted plans after a while.

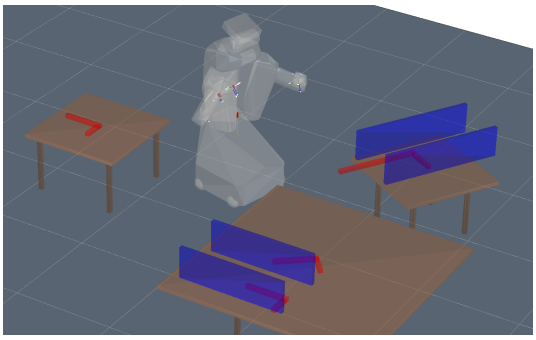


Fig. 7. Scenario 3: The target is behind the robot. Direct access would be possible, but the trajectory planner can't generate the required motion. Instead, it grabs another tool and, in letting go, takes a slightly different position.

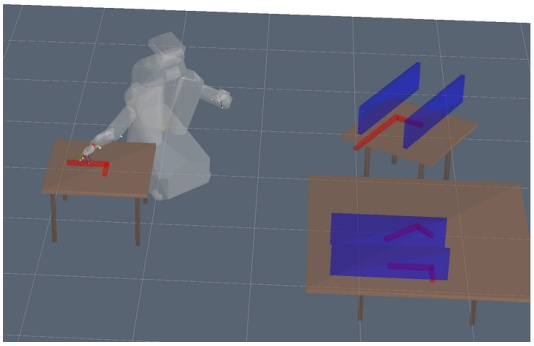


Fig. 8. From the new position, it is possible to sidestep just enough to reach the target.

In Scenario 4, once again, the implementation succeeded in very strange ways. The plans produced took up to a hundred steps, often repeating themselves and building tangles of connected tools. This, too, indicates problems using the MCTS planner; We eventually couldn't explore this further.

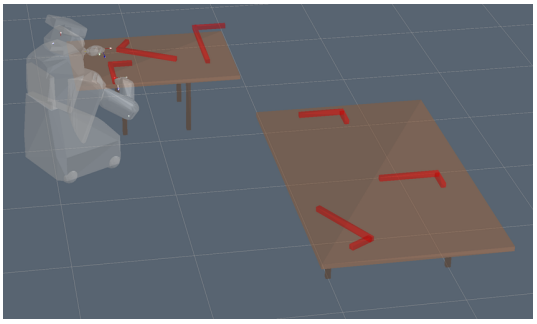


Fig. 9. Scenario 4: The target is on the right, not directly accessible but reachable with the help of one of the longer tools.

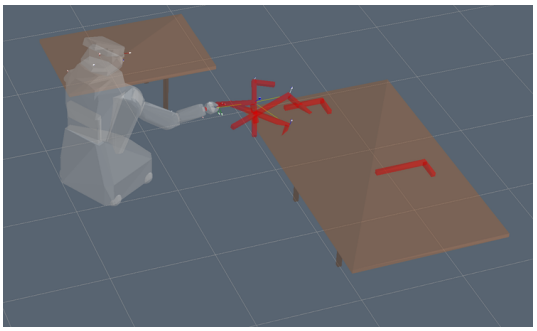


Fig. 10. The implementation generates a plan, but it is highly convoluted.

Overall, while the implementation does succeed in these toy problems, there is still a vast difference from it being able to solve sequential tool use scenarios in general. It is also important to note that even on this small scale, not all problems were solvable, owing to the limited function of the interface layer or problems with the trajectory planner.

7 Discussion

As stated in the opening section, this work has three components: one, posing in an explicit way a general task and motion planning problem; two, proposing clearly a highly separated way of approaching such problems; and three, applying (to a certain extent) this decomposition to the outlined problem of (meta-)tool use.

In the end, our implementation is able to find solutions to problems involving sequential tool use, but only in very small, highly limited toy scenarios. It is able to demonstrate that the approach taken can work, but it is far from a clean, general solution. Much work remains to be done on that problem. This is probably connected with the organization of this work: The theoretic part was done first, a lot of time being taken for getting a partial overview of the field and some first drafts. The implementation itself was not the sole center of attention and in fact was worked on mainly in the last third of the time. A different focus might have yielded more robust practical results.

It is also to note that we expect the problem to be approachable in most of the existing ways for TAMP, as well as probably others not yet thought of. The strict blackbox decomposition is but one way to do so; It had the benefit that we could use existing or supplied modules with little modification, only providing interfaces between them, but more tightly connected components might profit from better communication. One could supply the task planner with predicates evaluated on-demand in the geometric space, or one could merge the lower and interface level into a broader optimization problem, or many other possibilities. If a strict decomposition is attempted, there is room for improvement with the high-level planner that tends to generate needlessly complicated plans, with the low-level planner that appears to be too strict in preventing or too weak in allowing rotations and less direct motions, and with the interface layer that needs to do geometric backtracking or optimization.

References

BV04. Boyd, Stephen ; Vandenberghe, Lieven: *Convex optimization.* Cambridge university press, 2004

CAG09. Cambon, Stephane ; Alami, Rachid ; Gravot, Fabien: A hybrid approach to intricate motion, manipulation and task planning. In: *The International Journal of Robotics Research* 28 (2009), Nr. 1, S. 104–126

CHMS+. Chitnis, Rohan ; Hadfield-Menell, Dylan ; Srivastava, Siddharth ; Gupta, Abhishek ; Abbeel, Pieter: Learning an Interface to Improve Efficiency in Combined Task and Motion Planning.

DEK+12. Dornhege, Christian ; Eyerich, Patrick ; Keller, Thomas ; Trüg, Sebastian ; Brenner, Michael ; Nebel, Bernhard: Semantic attachments for domain-independent planning systems. In: *Towards Service Robots for Everyday Environments.* Springer, 2012, S. 99–115

EHP+11. Erdem, Esra ; Haspalamutgil, Kadir ; Palaz, Can ; Patoglu, Volkan ; Uras, Tansel: Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on* IEEE, 2011, S. 4575–4581

FN72. Fikes, Richard E. ; Nilsson, Nils J.: STRIPS: A new approach to the application of theorem proving to problem solving. In: *Artificial intelligence* 2 (1972), Nr. 3, S. 189–208

GLPK15. Garrett, Caelan R. ; Lozano-Perez, Tomas ; Kaelbling, Leslie P.: Ffrob: An efficient heuristic for task and motion planning. In: *Algorithmic Foundations of Robotics XI.* Springer, 2015, S. 179–195

LDB+14. Lagriffoul, Fabien ; Dimitrov, Dimitar ; Bidot, Julien ; Saffiotti, Alessandro ; Karlsson, Lars: Efficiently combining task and motion planning using geometric constraints. In: *The International Journal of Robotics Research* (2014), S. 0278364914545811

LDSK12. Lagriffoul, Fabien ; Dimitrov, Dimitar ; Saffiotti, Alessandro ; Karlsson, Lars: Constraint propagation on interval bounds for dealing with geometric backtracking. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on* IEEE, 2012, S. 957–964

LP15. Lertkultanon, Puttichai ; Pham, Quang-Cuong: A Single-Query Manipulation Planner. In: *arXiv preprint arXiv:1509.00600* (2015)

LPJM+89. Lozano-Perez, Tomas ; Jones, Joseph L. ; Mazer, Emmanuel ; O'donnell, Patrick u. a.: Task-level planning of pick-and-place robot motions. In: *Computer* 22 (1989), Nr. 3, S. 21–29

LPK14. Lozano-Perez, Tomas ; Kaelbling, Leslie P.: A constraint-based method for solving sequential manipulation planning problems. In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on* IEEE, 2014, S. 3684–3691

NW06. Nocedal, Jorge ; Wright, Stephen: *Numerical optimization.* Springer Science & Business Media, 2006

SFR+14. Srivastava, Sanjeev ; Fang, Eugene ; Riano, Lorenzo ; Chitnis, Rohan ; Russell, Stephen ; Abbeel, Pieter: Combined task and motion planning through an extensible planner-independent interface layer. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on* IEEE, 2014, S. 639–646

SLCS04. Siméon, Thierry ; Laumond, Jean-Paul ; Cortés, Juan ; Sahbani, Anis: Manipulation planning with probabilistic roadmaps. In: *The International Journal of Robotics Research* 23 (2004), Nr. 7-8, S. 729–746

SPGA13. Silva, Lavindra de ; Pandey, Amit K. ; Gharbi, Mamoun ; Alami, Rachid: Towards combining HTN planning and geometric task planning. In: *arXiv preprint arXiv:1307.1482* (2013)

THHG07. Taylor, Alex H. ; Hunt, Gavin R. ; Holzhaider, Jennifer C. ; Gray, Russell D.: Spontaneous metatool use by New Caledonian crows. In: *Current Biology* 17 (2007), Nr. 17, S. 1504–1507

Tou15. Toussaint, Marc: Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning. In: *IJCAI 2015*, 2015

VMAK10. Vasconcelos, Marco ; Monteiro, Tiago ; Aw, Justine ; Kacelnik, Alex: Choice in multi-alternative environments: a trial-by-trial implementation of the sequential choice model. In: *Behavioural processes* 84 (2010), Nr. 1, S. 435–439

WWC+09. Wimpenny, Joanna H. ; Weir, Alex A. ; Clayton, Lisa ; Rutz, Christian ; Kacelnik, Alex: Cognitive processes associated with sequential tool use in New Caledonian crows. In: *PLoS One* 4 (2009), Nr. 8, S. e6471–e6471

WWK11. Wimpenny, Joanna H. ; Weir, Alexander A. ; Kacelnik, Alex: New Caledonian crows use tools for non-foraging activities. In: *Animal cognition* 14 (2011), Nr. 3, S. 459–464