Combined Task and Motion Planning under Partial Observability: An Optimization based Approach

Camille Phiquepal **Marc Toussaint**
Machine Learning & Robotic Lab, University of Stuttgart
{firstname.surname}@ipvs.uni-stuttgart.de

not double blind? (I hope)

***Abstract*—We present new algorithms for Task and Motion Planning (TAMP) under partial observability. Our approach builds upon the Logic-Geometric-Programming approach (LGP) presentend in prior work, and extends the framework to handle partial observability. We model the problem as a particular kind of POMDPs where actions define trajectory cost and constraint functions. The numeric rewards, are unknown at first, but they will be gradually discovered by the motion planner when evaluating policies. The presented algorithm first assumes initial heuristic rewards and explores the policy space in an iterative process. Task Planning generates a candidate policy based on the last reward model using Value Iteration. Motion planning evaluates the candidate policy (solve the underlying constrained trajectory optimization problem) providing the real numeric reward value and the motion implied by the cost and constraints functions. Once the iterative process has reached an equilibrium (no more policy improvment), the motions of the best policy are reoptimized jointly as one single optimization problem. We evaluate our approach in simulation on object manipulation examples.**

are rewards really unknown?

maybe too technical for the abstract

no car examples?

I. Introduction

Robots must combine the ability to reason symbolically to take discrete actions (Task planning) and implement those actions in the real world i.e define paths / trajectories (Motion Planning). Integrated approaches for combining Task and Motion Planning are refered to in the litterature as Task and Motion Planning (TAMP). Most current TAMP research assumes full observability [1, 2, 3]. However partial observability is pervasive in many real world situations. In the field of Object Manipulation for example, when the environment is cluttered, object recognition may fail because objects are hidden or partially hidden. If some objects to manipulate are inside containers (objects inside a cabinet, a box, etc..), the objects may not be visible at first, and the robot has to be able to explore its environment to perform its task. The same kind of problematic arises for self driving cars that operate in a partially observable environment due to, for instance, the presence of other vehicles limiting the field of view of the ego vehicle.

except for Leslie & Tomas' work

In this paper we extend the Logic Geometric Programming approach (LGP) presented in prior work to partially observable environments. We model the task planning part as a POMDP and we plan assuming a start belief-state. The transition model and the observation model of the POMDP are assumed fixed. However, the reward values are updated during the search : Candidate policies generated by the task planning are given to the motion planner. The resulting trajectory costs are used to update the reward / cost model of the POMDP. From this point of view, our algorithm have strong affinity with model-based reinforcement learning.

has

Hmm. That might be confusing (planning vs RL..)

II. Related Work

III. Problem Statement

We model the Task planning part as a particular kind of partially observable markov decision process (POMDP) where trajectory cost and constraints functions are associated to each action. These functions link symbolic actions with continuous geometric motions. More formally, Let $\mathcal{X}$ be the configuration space of the whole environment, including the robot and all object configurations. We can explicate $\mathcal{X} = R^n \times SE(3)^m$ for an n-dimensional robot interacting with m objects.
Let $x : [t_k, t_{k+1}] \rightarrow \mathcal{X}$ be a path in the configuration space $\mathcal{X}$ of the whole environment in the time interval $[t_k, t_{k+1}]$.

To a given action $a \in A$, we associate the costs and constraints functions f_a, g_a, h_a. The motion cost of the action in the time interval $[t_k, t_{k+1}]$ is :

$$c(a,x) = \int_{t_k}^{t_{k+1}} f_a(x(t), \dot{x}(t), \ddot{x(t)})dt$$
$$s.t \quad g_a(x(t), \dot{x}(t), \ddot{x(t)}) <= 0$$
$$h_a(x(t), \dot{x}(t), \ddot{x(t)}) = 0$$

We define the reward of the motion x given the action a as the negative trajectory cost if the constraints are satisfied, minus infinity otherwise:

$$r(a,x) = \begin{cases} -c(a,x), & if\ g\ and\ h\ satisfied \\ -\infty, & otherwise \end{cases}$$

\text{...}

The utility (expected reward) of a motion x under a policy π starting from belief b_0 is defined as :

$$U_x^{\pi}(b_0) = \sum_{t=0}^{\infty} \gamma^t E(r(a_t, x_t)|b_0, \pi) \quad (1)$$

from t_k to t_{k+1}? or is x now a sequence of motion intervals?

The optimal joint TAMP policy Π is the symbolic policy (π^*) and motions (x^*) maximizing the utility :

$$\Pi^* = \{\pi^*, x^*\} = \underset{\pi,x}{\operatorname{argmax}}\, U_x^{\pi}(b_0) \quad (2)$$

are these a set of motion intervals?

The algorithms that we describe in the paper are ways to find solutions to this equation.

A. Decision graph

With the assumption that we plan from an initial belief state known at planning time, the set of all possible reachable belief states can be represented as a graph. Each node is a belief state. There are two kinds of nodes :

- Actions nodes : the agent has to choose which action to take. The edges starting from the node are the different possible actions, a reward is associated to each action-edge : this is the reward received after executing the action
- Observation nodes : the agent receives an observation, each edge starting from an observation node is a possible observation.

1) Example of decision graph on a toy example: Let's consider a robot in front of two boxes. The robot has to grasp a ball which is in one of the two boxes. The robot can open the boxes and recognize if the ball is in a box or not. The Figure 1 is the decision graph of this simple model. The rectangular nodes are observation nodes. The initial belief state is colored blue. The green nodes are termial belief states.

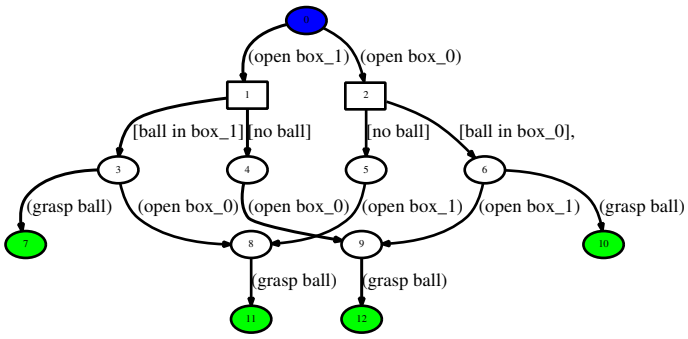


Fig. 1: Example of decision graph

2) Policy: A policy is a mapping from belief states to actions. It is a subset of the decision graph, each action node has one single child. Optimizing a policy consists in determining the best action at each action node. The thick edges on the Figure 1 represents a possible policy. The decision graph potentially contains cycles, however, under the assumption, that trajectory costs are always stricly positive, the optimal policy is assured to be a tree (no cycles). In the case of full observability, the policy boils down to a sequence of actions.

IV. POMTP Algorithm

Our approach for optimizing a TAMP policy (solve the equation (2)) is schematized on the Figure 2. First, the decision graph is built with heuristic reward values. Secondly, several iterations of Task Planning and Fast Motion Planning are performed : Task planning computes a candidate policy based on the last up-to-date reward model. The policy is given to the motion planner which will compute the trajectories and costs of each action. The resulting costs are used to update the rewards associated to the action-edges. Task planning is re-run with the updated reward model which potentially results

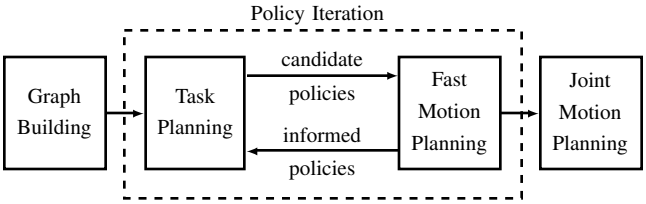


Fig. 2: POMTP algorithm

in a new policy candidate. This process is iterated until an equilibrium is reached (no more policy improvement). Finaly, a pass of joint trajectory optimization is performed on the best policy found. This pass doesn't plan each action trajectory in isolation but optimizes the trajectories as a whole. It is typically way slower but gives smoother and more optimal results. Different optimization parameters (more time steps) can be used in this final stage. The next sections detail each part of the described algorithm.

The initial reward values used when building and initializing the graph (without any information coming from the motion planning) are crucial to control the exploratory behavior of the optimization and whether the equilibrium is reached at a globall optimal policy or not. The section IV-E discusses the influence of the initial rewards.

A. Graph Building

The decision graph, is expanded from the start belief state using a breadth first strategy. In the general case, the number of reachable belief states is infinite leading to an inifinite decision graph. The algorithms that we present in this paper assume a graph of finite size. To constraint the graph size to be finite, one solution is to expand the graph up to a certain maximal depth. In this case, the graph can represent only a subset of all the possible policies. Under specific assumptions, the decision graph is also guaranteed to be finite without having to restrict the depth of the graph expansion. This is the case if both the transition model and the observation model are deterministic. Under such an assumption, the agent is still faced at with uncertainty due to partial observability.

B. Task Planning on the decision graph

Under the optimal policy, the value of each action node obeys the following Bellman equation :

$$V^{\star}(b) = \max_{a} \left[R(a,b) + \gamma \sum_{o \in \Omega} O(o|b,a) V^{\star}(succ(b,a,o)) \right] \quad (3)$$

In the above expression, $succ(b,a,o)$ is the successor belief state of b after receiving the observation o. In our toy example (1), $succ(node_0, (open\ box_1), [ball\ in\ box_1]) = node_3$. To find the optimal policy, we compute the value of each action node using Value iteration. At each pass, the algorithm goes through each action node and updates its current value based on the values of the children node. The update process is iterated

until the values are stable i.e. the Bellman equilibrium has been reached.

$$V^{\star}_{i+1}(b) \leftarrow \max_{a} \Big[R(a,b) + \gamma \sum_{o \in O(b,a)} O(o|b,a) V^{\star}_{i}(succ(b,a,o))\Big] \tag{4}$$

Once the value is know, the optimal policy is retrieved by selecting at each action node the action leading to the children with the highest expected value.

C. Fast Motion Planning

Task Planing gives candidate policies to the Motion Planner. The execution time of this phase is crucial for the overall planning time, because it is done multiple times. This is perfomed in two steps, a first feasibility check and then a trajectory optimization. For these two steps, each action is optimized indenpendantly in a breadth-first order. Going back to the simple example of the policiy on the Figure 1, the results of the optimization of the action $(open\ box_1)$ between the nodes 0 and 1 define the start configuration for the optimization of the actions $(grasp\ ball)$ between the nodes 3 and 7 and for the action$(open\ box_1)$ between the nodes 4 and 9. Trajectory Optimization is performed using the Logic Geometric Programming framework (LGP) presented in previous work[X]. An action edge is only planned one time. There may be a strong overlap between candidate policies (same edge in many candidate policies). This is especially the case in the last iterations of Policy improvement, and for edges close to the start node, but Motion Planning is performed only on the edges that haven't been planned yet. Intuitively, as Policy iteration goes along, the decision graph is filled with geometric information. The initial reward is the parameter which controls how large the coverage will be and how fast convergence occurs. Early stopping and re-using results of previous optimizations for the planning of new policies are ways to speed-up the optimization and the reason why we call it Fast Motion Planning.

1) Early stopping : Pose level optimization: To detect quickly which trajectories are infeasible, we first optimize key-frames only (robot pose at each node). This step is much quicker than optimizing a trajectory. If an action is impossible during the feasibility check, the optimization is not pursued further. This feasability check is optimistic, in other words, it might succeed even if the path itself is infeasible (no possible trajectory without collision between two keyframes for example). Consequently, this initial step is a good way to detect early if an action is infeasible whitout excluding potential feasible policies. This gives a feasibility information but doesn't provide with trajetory costs.

2) Trajectory optimization: The second pass of optimization consists in optimizing between key-frames, we consider typically 20 time-steps for each action. In addition to the costs and constraints functions defined by the action, the robot dynamic and collision avoidance are considered which results in accurate trajectory costs that will be used to udpate the reward model of the decision graph.

D. Backup Mechanism

The initial rewards of decision graph are replaced with the trajectory costs resulting from the Fast Motion Planning. The resulting geometric configuration of the robot at the end of each planned action is also saved. If an action is infeasible, the reward is set to minus infinity which excludes this edge from next candidate policy.

E. Rewards initialization and graph exploration

The initial rewards influence drastically the search behavior. Optimistic initial rewards encourage exploration. This can be intuitively understood, since unexplored edges have big rewards, the Value iteration will tend to converge to a policy having unexplored edges. On the other hand, with pesimistic initial rewards, the equilibrium is attained after a first policy has been successfully optimized by the motion planner. Of course, although such a policy is feasible it is less optimal than the one got with a bigger initial reward.

F. Joint Motion Planning

Optimizing each action independantly, has one drawback, it can't capture long-term effects on the trajectories. When considering a policy as one single optimization problem, final actions potentially influence motions earlier on the trajectory-tree. Moreover, planning parameters (e.g. number of key frames) may be different when planning for informing the search or planning for outputting the final policy. This is why we introduce a final stage of optimization for reaching a better optimum and allows smoother motions. This typically takes much longer but this is done on one single policy.
Optimizing the trajectory tree as a whole is not straighforward because of the branching factor. We solve it in two phases : First, linear trajectories (without branching points) are optimized from the root graph node to each one of the terminal belief states (see opt_1 and opt_2 on the Figure 3). Secondly, the trajectories are re-optimized with additionnal constraints enforcing that the common parts between trajectories are identical (see opt_3 and opt_4). The re-optimizaton is potentially performed multiple times until the equality constraints across paths are fully satisfied (in practice, one iteration is often enough).

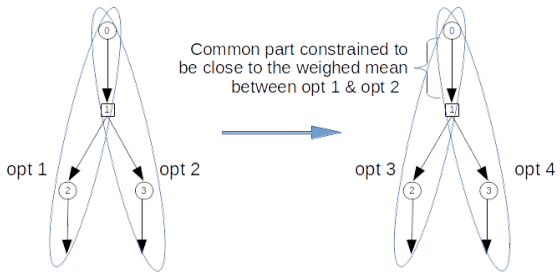
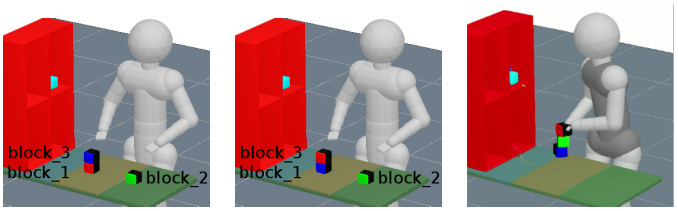


Fig. 3: Joint optimization

Ha, this is a form of ADMM! We should emphasize this. Sounds good and really makes sense, and there is lot's of theory on it...



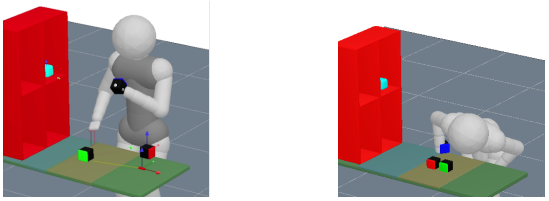
(a) first start configuration with model B

(b) second start configuration with model B

(c) example of goal configuration

Fig. 4: Example of start and goal configurations

mention that the colors are not visible to the robot from behind?



(a) *Look* with the block in hand

(b) *Look* just after having posed a block

Fig. 5: Example of geometric configurations for the *Look* action

V. Experiments : Sussman anomaly under partial observability

We consider an humanoid robot (see Figure 4). There are three blocks a table. The robot has to stack the blocks in a given order on one of the three table locations (red on the top, green in the middle, blue at the bottom). We compute trajectories for all the robot joints (27 degrees of freedom). Only the robot left hand is assumed to be able to grasp blocks. The blocks are black with one colored side which identifies it. This side is assumed to be always be the opposite side of the block. The robot knows where the blocks are (refered as *block*_1, *block*_2, *block*_3). However the robot can't see the colored side at first and has to perform exploration to identify the blocks and to build the stack in the correct order. The table is subdivised in 3 different locations (table left, center and right), if a block is already at a location, the location is considered occupied (no block can be added on the same table part). There are 3 possible actions:

- **Look** a block : the robot seeks to align its sensor (robot head) with the colored side of the box. This will typically lead the robot to both move its head and its hand simultaneously (see 5a). After this action, the agent receives an observation (color of the block).
- **Grasp** a block : This destroys the link between the block and its supporting object which enables the agent to move the block.
- **Place** a block at a location : the block is placed at the given table location, or onto another block.

We evalutate two slighlty different action model definitions. In the first variation (Variation 1), the action *Look* has a symbolic precondition : the robot should be holding the block before checking a block. In the second variation (Variation 2), this precondition is removed. The *Look* action is possible much more often which increases the branching factor and the size of the overall decision graph. However, most of the time, when the robot doesn't have the block in hand before checking, the action is infeasible geometrically : Indeed the robot has to place place its head far ahead and look backwards which is, in most cases, infeasible given the geometrical constraints of the robot. It is however interesting to note, that the *Look* action without grasping is possible in some cases (if the robot has just previously placed the block close to the table border with some orientation see Figure 5b). In other words, the grasp-precondition is not absolutely necessary to ensure the feasibility of the *Look* action. Some candidate policies generated are successfully planned but usually lead to high costs and are not retained as the optimal policy. This second variation is used to analyse how our approach works with a more "free" symbolic problem description causing motion planning failures.

To evaluate the scalability of the approach, we test with 3 different initial belief states configurations. In the first configuration (A), the agent has a prior knowledge of the color of each block. This boils down to the fully observable case since there are no relevant unobserved parameters. To keep the action model unchanged, we still impose that the agent has to look one block to complete its task. In the second configuration (B), 2 blocks are unknown. The figures 4a and 4b show the two possible start configurations with this model. In the third case (C), there are no prior knowledge which leads to 6 possible initial configurations. The different start configurations are summed up in the table V. In all cases, the initial belief state is uniform (e.g. 1/2 likelihood for each possible start configuration with the model B, 1/6 likelihood for model C).

Problem variation	Description
A-1	3/3 blocks known; grasp before looking
A-2	3/3 blocks known; no grasp precondition
B-1	1/3 blocks known (block_2); grasp before looking
B-2	1/3 blocks known (block_2); no grasp precondition
C-1	0/3 blocks known; grasp before looking
C-2	0/3 blocks known; no grasp precondition

	Belief state size	Graph size	Graph building time
A-1	1	192	0.20
A-2	1	192	0.24
B-1	2	336	0.53
B-2	2	480	1.08
C-1	6	2076	8.55
C-2	6	4128	34.3

The figure 6 shows an optimized policy for the model B-1 and B-2. The agent first grasps a block and looks it, there are two possible outcomes. Once the block has been identified,

the agent pursues the stacking.

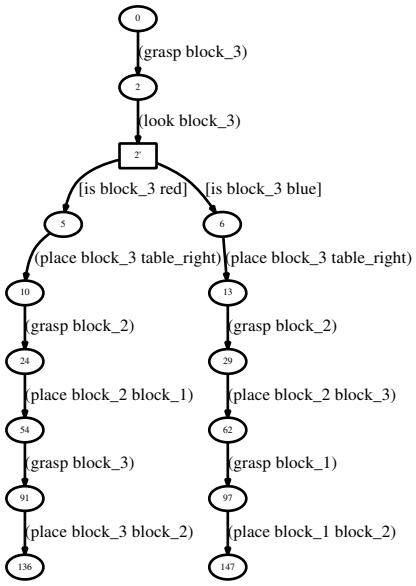
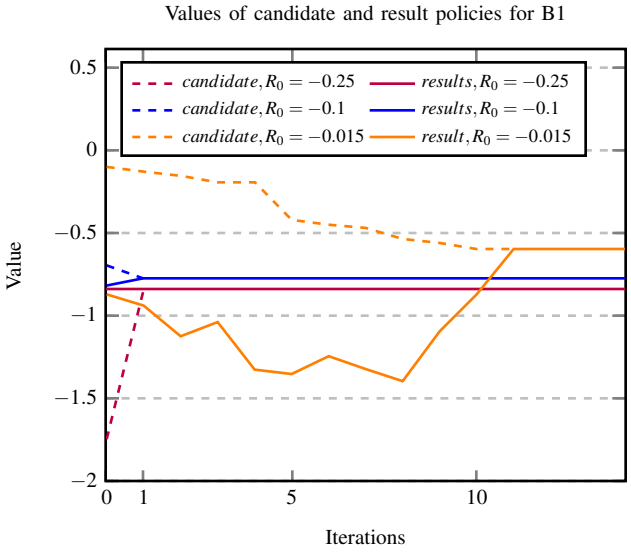
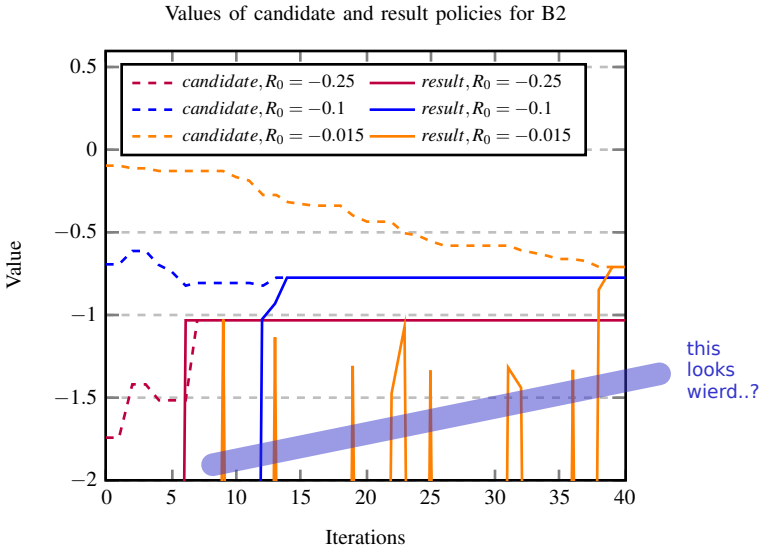


Fig. 6: result policy for B2

	R_0	Iter-ations	N of actions*	Task planning	Fast motion planning	Joint motion planning	Total*
A-1	-0.25	1	7	0.007	1.37	11.4	13.0
	-0.1	2	7	0.018	3.14	9.7	13.1
	-0.015	13	7	0.10	18.5	16.3	35.1
A-2	-0.25	2	7	0.013	1.81	8.43	10.5
	-0.1	3	7	0.019	2.08	11.6	14.0
	-0.015	16	7	0.10	14.8	11.4	26.6
B-1	-0.25	1	12	0.014	2.72	26.2	29.5
	-0.1	1	12	0.016	2.85	21.5	24.9
	-0.015	11	12	0.13	30.6	20.6	51.9
B-2	-0.25	7	12	0.089	9.10	56.7	66.9
	-0.1	14	12	0.17	20.4	59.9	81.4
	-0.015	39	12	0.55	69.9	38.6	110.1
C-1	-0.25	1	33	0.077	11.6	172.3	192.5
	-0.1	8	33	0.35	56.9	105.5	170.9
	-0.015	41	37	2.07	321.3	112.7	444.1
C-2	-0.25	15	33	1.16	42.4	100.8	182.8
	-0.1	48	33	3.38	146.5	250.9	436.6
	-0.015	303	39	26.0	1188.9	261.5	1510.9

* Number of actions of the final policy
** The total planning time also includes the graph building time given in the table V





1) Influence of initial rewards: With an initial reward of -0.25 (pessimistic initial reward), the search finishes as soon as a first policy is found. This happens after one single iteration with the B-1. With the B-2, the search encounters infeasible actions, the first possible policy is found at the 7th iteration. This policy is less optimal than the results found with the other initial rewards.

The highest initial reward (-0.015) is always optimistic. Candidate policie values (containing at least one unexplored action) are consequently always higher than the result policies (see the orange curves on the figure V and V. This leads to the most exploratory behavior : The Value iteration pass in Task Planning tends to converge to policies containing unexplored actions (still having R_0 as hypothetised reward). It requires the biggest number of iterations. In particular, Task Planning will also explore deeper candidate policies (with more action steps). In some cases a policy with slightly more actions results in a better trajectory cost (see B-2 with R_0 = -0.015). The search finally converges to a better policy than with the other initial reward values. In this example, the small improvements occuring in the last iterations are mostly due

to small rearrangements of the target location when placing blocks (e.g. place a block on table-left instead of table-center).

2) Influence of the action precondition: Removing the precondition increases strongly the decision graph size, more Task-Planning iterations are needed to reaching an equilibrium, and it leads to the generation of candidate policies that are infeasible geometrically. This is particularly visible on the figure V, the curves of result policies are very discontinuous because the majority of them are infeasible. When motion planning fails, this is, most of the time, due to one single action that couldn't be planned. The policy value is minus infinity, but the resulting rewards of each possible actions still informs the decision graph leading to an overall improvment. The search reaches an optimal policy which is as good as the policy obtained with the model with pre-condition. We think that this is an important quality of the proposed solution. Adding domain specific knowledge in the Task Planning (to ensure that motion planning will succeed) helps to speed up the search. However, in the general case, we think that it is not always possible / convenient to incorporate such geometric reasoning (reachability of a view point, reachibility of an object) in the logical reasoning.

3) Execution time and scalability: The overall planning time is dominated by the motion planning for all models. As long as the model is simple (A or B) or the exploration kept low (R_0 = -0.25), Motion Planning is dominated by the single pass of joint global optimization. The execution time of this single pass mainly depends on the total number of action steps in the policiy and on the belief state size but is indenpendant from the number of iterations occuring before. In the configurations requiring the biggest number of iterations (C-1 and C-2 with R_0 = -0.015), motion planning and the overall planning time are dominated by the stepwise motion planning phases of the policy improvment iterations. It also appears clearly that scalability is an interesting problematic here. All the parameters of the problem increase drastically with the size of the belief state (graph size, required number of iterations, size of the resulting policy, computation time). Parametrizing the search with a very exploratory behavior (high R_0 may be feasible for problems of small sizes but suffers from the curse of dimensionality. One way to still enable some exploration while maintaining a bounded planning time is be to always save the best candidate policy successfully planned so far and interrupt the policy optimization iterations when a given time limit is reached. In the next paragraph we explain some future directions we would like to explore to improve performances.

VI. Conclusion & Future Work

We proposed a new, optimization-based approach to manipulation problems. This approach handles partial observability by reasonning over the agent + environment belief state and by optimizing trajectory-trees that can account for the observation branching. It can plan policies that combine both exploratory actions (mostly sensor trajectories) and exploitative actions (e.g grasp, place). The deegree of exploration over the vast space of all possible manipulations policies can be controlled by one single parameter (initial heuristic reward). As motion optimization is time consuming, the ability to quickly detect if an an action is infeasible is crucial and we perform it by performing a fast pose-level optimization. Moreover, the policy iteration process naturally copes with motion planning failures that are not handled as errors but simply inform the decision graph with the real cost (albeit infinitely big) of a given action. Scalability becomes an issue when the number of manipulations and or the size of the belief state increases. An efficient way to decrease the complexity and speed-up the search is to have a task-level model that is accurately tailored for the problem to solve (example of the grasp precondition), this prevents too many motion planning failures and limits the branching factor. We intend to explore, in future work, how the task-level model can be refined and learned using the results from multiple planning queries.

Our current method computes policies that adress every possible outcome during the possibility execution. To scale better to problems having a larger belief state size, we would like to investiguate an approach where the policy is planned only for handling the most probable belief-state trajectories to limit the complexity. As such a policy can't handle every outcome at execution time, replanning would be triggered once the execution layer detects that the system is evolving toward a belief state which is not covered by the current plan.

Finally, we would like to apply the current framework to the field of autonomous driving by considering driving maneuvers as symbolic actions.

References

[1] M. Toussaint and M. Lopes, Multi-Bound Tree Search for Logic-Geometric Programming in Cooperative Manipulation Domains. Accepted at ICRA 2017.

[2] M. Toussaint, Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning. In Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI 2015), 2015.

[3] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, Incremental Task and Motion Planning: A Constraint-Based Approach, in Robotics: Science and Systems, 2016

[4] Leslie Pack Kaelbling, Tomas Lozano-Perez. Integrated Task and Motion Planning in Belief Space, International Journal of Robotics Research, 2013