Pick and Place Task & Motion Planning: LGP vs. PDDLStream

Christina Wettersten
6.881
Massachusetts Institute of Technology
Cambridge, MA USA
cwetters@mit.edu

***Abstract*—The robotic task of picking and placing items requires a planner to make sense of high dimensional, complex relationships on continuous variables as well as logic based reasoning over discrete variables. Recently there have been two attempts to try and solve these two parts of the task jointly: PDDLStream and LGP [1]. PDDLStream [2] approaches by extending the existing discrete logic solving of PDDL to the the many collision, kinematic and motion constraints in robot trajectories and object poses. From the other side, LGP attempts to add some logical reasoning to the approaches based in constraint optimization and Mixed Integer Programming. We will compare how these two approaches work in a simple pick and place setup.**

***Index Terms*—LGP, PDDLStream, Kuka**

I. Introduction

Traditionally robotic tasks have been divided into two categories: planning and optimization. For some problem spaces this separation of the unbounded logical ordering of actions from the bounded optimization of subtasks makes otherwise intractable problems solvable. For example, consider a fleet of UAVs in open areas delivering goods. The individual sections of the flight paths have little to do with the larger logic problem of which drone delivers where.

However with a robotic task like picking and placing items, the movement and positions of the robot objects are complexly related. If the planner first wants to place an item on shelf it needs to ensure that it not only satisfies the logical constraints of having the object and an empty place to put it, but also that there is a feasible motion path to move both the robot and the object around the other obstacles.

Connecting these two layers has previously been done by specialized procedures adapted for specific problem domains. While these can preform well for their given tasks, they lack in generalizability. Recently two approaches have come out to attempt to solve problems involving both discrete logic and continuous optimization: LGP and PDDLStream.

Starting from PDDL(Planning Domain Description Language), a way to describe discrete logic problems, PDDLStream adds in streams which act as an interface to sample based procedures to solve continuous variable problems like motion planning.

Working from the other side, LGP (Logic-Geometric Program) is expanded off of Hybrid Control, Contact Invariant Optimization and MIP (Mixed Integer Programs). By describing a set of differentiable physical relations between objects and the start and end of actions, LGP allows for the formulation of optimization problems based on discrete logic.

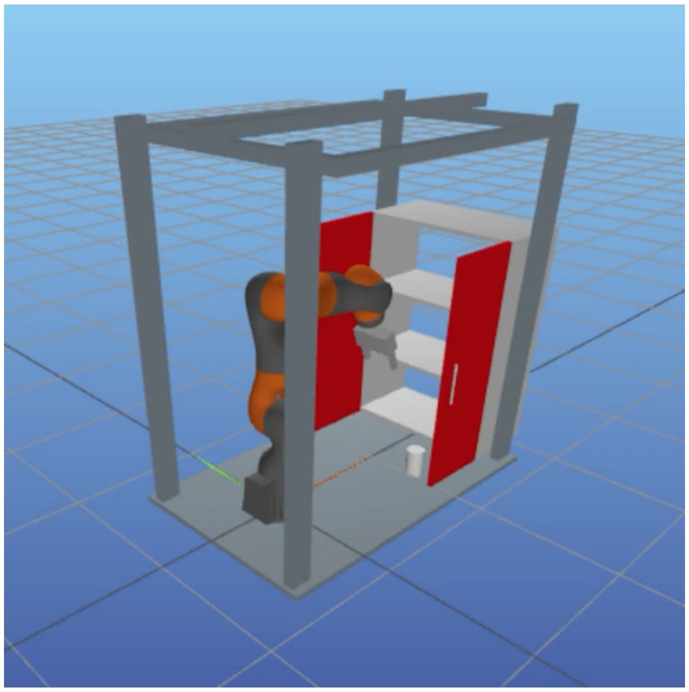


Fig. 1. The simulated environment for PDDLStream using Drake

The approach here follows the work by Toussaint et. al on LGP [1] and Garrett et. al on PDDLStream [2] applied on simple pick and place setup using a Kuka arm. The tasks explored involve picking and placing objects in both clear and cluttered environments in and around a cabinet.

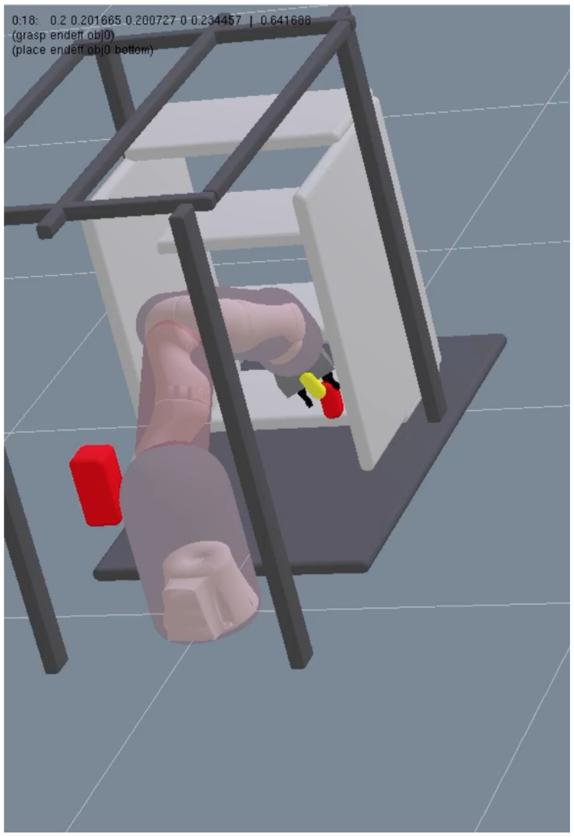
II. Related Work

Other attempts from the TAMP side (Task And Motion Planning) at robotic pick and place tasks involve breaking the continuous robot space into discrete sections so that it can be reformulated as a Constraint Satisfaction Problem (CSP) [7] [9] or using a search over sequences formed by the task planner, paired with a standard sample based path planner proving individual actions improbable [10] [11].

However, neither fully incorporates the continuous dynamics of the workspace in the interplay between the task and motion planners, and those able to plan for realistic robotic domains have required specialization for that class of problems.

One way motion planners solved a similar issue of complex sequential movements, like walking, was through multi-modal planning [5]. This used the concept of modes, defined by sections of a path where the set of objects or surfaces a robot is in contact with remains constant. For problems like walking, the structure and order of the contacts are well known and can be explicitly described [4]. The problems can be formulated as Mixed-Integer Programs (MIP) that can be solved with standard branch-and-bound optimizers.

Not all problems are as well understood. Contact Invariant Optimization (CIO) [8] searches over possible combinations of surfaces to create phases and the possible phases to create full movements. It has been used successfully to preform complex robot movements like moving from sitting to standing or crawling, but has not yet been shown to achieve higher level task planning.



```
0:18:   0.2 0.201665 0.200727 0 0.234457  |  0.641688
(grasp endeff obj0)
(place endeff obj0 bottom)
```

Fig. 2. The simulated environment for LGP using RAI

III. LGP

Toussaint et al. [1] proposed a way to apply the dynamic based flexibility of CIO to the logically structured search in TAMP to solve sequential manipulation tasks involving tool use. The approach uses a notion of modes, similar to CIO, where the equations of motion describing the objects are constant.

In [1] the problem is formally stated as taking the configuration space $X = R^n \times SE(3)^m$with a n-dimensional robot and m rigid objects with initial configuration $x_0 \in X$. The aim is to find a path

$$x : [0,T] \to X$$

$$\min_x \int_0^T f_{path}(\bar{x}(t))dt + f_{goal}(x(T))$$

s.t.

$$x(0) = x_0, h_{goal}(x(T)) = 0, g_{goal}(x(T)) \leq 0,$$

$$\forall_{t\in[0,T]} h_{path}(\bar{x}(t)) = 0, g_{path}(\bar{x}(t)) \leq 0. \tag{1}$$

The path constraints are in terms of $\bar{x}(t) = (x(t), \dot{x}(t), \ddot{x}(t))$ and describe the kinematic limitations, and $(f, h, g)_{goal}$ define the objectives or constraints on the final configuration. Drawing inspiration from CIO, the problem can be simplified by assuming the paths are composed of phases of modes. Following [1], the problem can be reformed as a Logic-Geometric Program (LGP) [12].

$$\min_{x, a_{1:K}, s_{1:K}} \int_0^T f_{path}(\bar{x}(t))dt + f_{goal}(x(T))$$

s.t.

$$x(0) = x_0, h_{goal}(x(T)) = 0, g_{goal}(x(T)) \leq 0,$$

$$\forall_{t\in[0,T]} h_{path}(\bar{x}(t)|s_{k(t)}) = 0$$

$$\forall_{t\in[0,T]} g_{path}(\bar{x}(t)|s_{k(t)}) \leq 0$$

$$\forall_{k=1}^{K} h_{switch}(\hat{x}(t)|a_k) = 0$$

$$\forall_{k=1}^{K} g_{switch}(\hat{x}(t)|a_k) \leq 0$$

$$\forall_{k=1}^{K} sk \in succ(s_{k-1}, a_k) \tag{2}$$

Previous LGP formulation [12] only addressed kinematic reasoning. By defining path constraints in terms of $\bar{x}$ instead of $(x, \dot{x})$ and switch constraints by explicitly referring to discrete action decisions a_k instead of pairs of consecutive modes (s_{k-1}, s_k), the LGP can also allow for physical reasoning [1].

The LGP can be easily compared to the MIP used in hybrid systems [4]. Instead of integers it uses a first order logic state s_k to indicate the mode which allows the high level task planner in TAMP to be tied to the modes in the path optimization. The existing LGP solver [6] is closely related to the MIP branch-and-bound, but applies several bound approximations for better computational efficiency.

As laid out in [1], the LGP is solved using a Multi-Bound Tree Search (MBTS) which searches over a decision tree described by the sequences of actions $a_{1:K}$ or 'skeletons'. Each node on the decision tree represents a possible skeleton and its corresponding path optimization problem $P(a_{1:K})$ which solves for the overall optimal manipulation path given the skeleton of actions. As $P(a_{1:K})$ is an expensive non-linear

program, the MBTS is guided by cheaper, simplified versions looking at coarser time discretizations of the path.
The approach uses two types of modes: kinematic and stable modes. The kinematic modes based off the definitions of modes by Hauser [5] in the physical domain align with the contact modes in CIO [8]. Stable modes are defined to be phases of the path which require the relative transformation between objects to remain constant. This naturally includes objects resting on top of each other or firmly grasped, but also pushes- so long as they are stable. It doesn't include objects slipping or rotating relative to one another [1].

The approach then uses predicates and logical rules to describe possible sequences of mixed stable and kinematic modes- allowing for some objects to be defined by dynamics and others by stable kinematic relations. The predicates used describe the physical relations that can exist between objects, e.g. [touch X Y], and the dynamic or kinematic constraints they impose on path optimization. For the non-persistent predicate touch, it imposes the constraint GJK between X and Y to be 0. The geometric predicates allow the definition of geometric constraints necessary to transition between modes.

The action operators then describe the different ways to control the predicates. For example, a hit would imply the predicate [touch X Y], a predicate for impulse exchange from X to Y, [impulse X Y] and the creation of dynamic free joint for the object Y, (dynFree Y).

For a full description of predicates and actions see the original paper [1].

IV. PDDLStream

PDDLStream is an implementation of STRIPStream [2] using PDDL to describe actions and streams. It extends off of the STRIPS language [3]. A STRIPS problem is given by (A, I, G) which define a set of actions A, an initial sate I made up of literals, and a set of goals G. Each action a is defined by the set of preconditions required $pre(a)$ and effects $eff(a)$ the action applies.

An action a is *applicable* in state S if the preconditions are met $(pre^+(a(\bar{x})) \subseteq S) \wedge (pre^-(a(\bar{x})) \cap S = 0)$. Here $+, -$ denote the positive and negated literals and $\bar{x}$ denotes a specific set of objects. For example 'cup not in hand' would be a negated literal and object tuple $\bar{x}$ would include the instance of an object 'cup'. Applying an action creates a new state S' $(S \cup eff^+(a(\bar{x}))) \setminus eff^-(a(\bar{x}))$.

The solution of a STRIPS problem is a legal plan $\pi = \langle a_1(\bar{x}_1), ..., a_k(\bar{x}_k)\rangle$ made of k actions where each action $a_i(\bar{x}_i)$ in the sequence is applicable in the state resulting from applying the previous action and the first action is applicable in the initial state I. To describe the legality of the plan with respect to the initial state, a *preimage* of a plan π is defined as a set of literals needed in I in order to execute π.

$$PREIMAGE(\pi) = \bigcup_{i=1}^{k} (pre(a_i(\bar{x}_i)) - \bigcup_{j<i} eff(a_j(\bar{x}_j)))$$

As described in [2], a STRIPStream problem is given by (A, S, I, G) where A, I, and G are given as before. The new S denotes a set of Streams. A *stream* s is a type of conditional generator $f(\bar{X})$ with a specific domain, range and graph. Conditional generators turn an object tuple $\bar{x}$ into a generator $g_{\bar{x}} = \langle \bar{y}_1, \bar{y}_2, ...\rangle$ which is a finite or infinite sequence of object tuples $\bar{y}_i$. The conditional generator is implemented as a blackbox procedure which specifies the binary relation of inputs $\bar{x}$ and outputs $\bar{y}$.

A stream s can be described by its $dom(s)$ and $cert(s)$. $dom(s)$ represents the domain and specifies the sets of object tuples $\bar{x}$ for with s is defined, and $cert(s)$ denotes a set of *certified* atoms on the input and output ($inp(s)$, $out(s)$) parameters, representing the range and graph of s.

STRIPStream problems are restricted to problems with static predicates, and requires that atoms certified by streams are not negated for any action's precondition. This allows the atoms certified by streams to be considered constant. The initial state I is then augmented by these constant facts from the set of streams S:

$$I_S = I \cup \{p(\bar{x}, \bar{y}); s \in S, |\bar{x}| = |inp(s)|,$$
$$dom(s(\bar{s})) \subset I_S, \bar{y} \in s(\bar{x}), p \in cert(s)\}$$

The solution for a STRIPStream problem is a legal plan π where $PREIMAGE(\langle \pi, G\rangle) \subset I_S$.

For more detail and links to related proofs see the original paper [2].

V. Problem Formulation and Setup

The problem domain explored here involves robotic picking and placing. The setup included a Kuka arm, a small tabletop work area, a simple double doored cabinet and a variety of objects for the robot to move. The tasks were defined such that the cabinet and objects were within the robot's reachable work area.

While the same hardware, objects and physical station are used to run plans generated by both planners, it should be noted that each planner defined its own environment and robot model file including the definition of collision/contact regions. When running on the hardware, outputs from both planners were produced in the form of 8-dof waypoints specifying the 7 joint angles of the kuka arm and the 1-dof of the gripper.

VI. Early Results

In order to test the two planners against each other a simple simulation environment was constructed. While efforts were made to ensure the simulated environments were identical, the differences in model description languages made it a bit complex to convert from one to the other and there were slight differences. As the planners are inherently different, the logical actions and their respective definitions were matched as best as possible.The two planners were tested with a variety of simple tasks involving grasping an object on the table and putting it on a shelf in the cabinet.

With no other objects in the scene, both solvers could reliably find solutions for any shelf. LGP typically found a solution for this simple problem in <1 second where PDDLStream was a bit more variable but generally finding a path in around 2 seconds.

Keeping the goal the same (place one object on a specific shelf), as more objects are added the LGP planner took longer and longer to find a solution. With 2 objects it took 2 seconds, with 4 objects around 15 seconds, with 10 objects it took over a minute. However, even if it was slow to compute a solution was generally found. PDDLStream had different results with plans (if they were found) being found in relatively short time, but sometimes it returned that the problem was infeasible. With 2 objects PDDLStream took 2s and found a path 82% of the time. With 4 objects around 4s, success rate 66% and at 10 objects no plans were found.

The path length for LGP plans was on average 2.5x shorter than a plan generated by PDDLStream.

VII. Limitations

PDDLStream and LGP have different strengths which are visible even from the early results. Where LGP can move in tightly constrained spaces with objects packed in close together, as the number of objects grows the planning time rises quite quickly. For this specific setup however, as the tabletop is fairly small, the limitations of LGP are inline with the physical limitations of what we can place in the scene. This makes LGP a surprisingly good planner for the setup.

PDDLStream on the other hand, can handle the larger number of objects, but struggles when the objects are near to each other. In a tightly constrained space like this setup PDDLStream, which uses very loopy, sample based paths to get from one point to the next, struggles to find clear paths with even a relatively small number of objects. In a more open environment however, I would expect PDDLStream to start outpacing LGP as the number of objects increase.

In running on real hardware, the small differences in the robot/station models made it difficult to directly compare the paths. The LGP plans which were highly optimized targeted placing items on the very edge of shelves, which made them susceptible to just missing the shelf. The PDDLStream plans because they moved through so much of the configuration space to get between points took longer to execute and consistently went closer to the edges of workable space, occasionally hitting the stations self-imposed joint and end effector position limits.

VIII. Future Work

Any more rigorous comparison between the two models would benefit from a stricter definition of problem and model space. To really compare the two plans, the model files and logical actions available should be more closely matched. Both model languages have similar feature sets and interpreter from one to the other would be the first task to accomplish.

Exploring other robot models would also highlight the comparative strengths of the two planners, the pr2 is shown in both original research papers and would be an easy next model to consider. Specifically a more open domain where the robot could approach an object from multiple directions would be good to contrast against this very confined domain.

Another avenue to consider would be adding actions other than pick and place to this problem setup. Opening cabinet doors would be the most obvious next action to add. But other small tabletop constrained actions like pushing could be added as well.

IX. Implementation and Videos

RAI and Ry (the python bindings for RAI) were used as the LGP solver. The code can be found:https://github.com/MarcToussaint/rai-python. The base PPDLStream planner to solve PDDLStream problems can be found here:https://github.com/caelan/pddlstream. The robot station setup for execution on hardware was based off of Drake and can be found here:https://github.com/RobotLocomotion/6-881-examples.

The modified robot and model files for the tests outlined in this paper, and videos of the path planning generating a variety of good and bad paths can be found at https://github.com/cwetters/lgpvpddl. Development was mainly done in a jupyter notebook running Python 3.

References

[1] M. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum: Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning. *In Proc. of Robotics: Science and Systems (R:SS 2018), 2018.*

[2] Caelan R. Garrett, Tomas Lozano-Perez, Leslie P. Kaelbling. STRIPStream: Integrating Symbolic Planners and Blackbox Samplers, 2018.

[3] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving.*Artificial Intelligence*,2:189–208, 1971.

[4] Robin Deits and Russ Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. *In Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference On*, pages 279-286. IEEE, 2014.

[5] Kris Hauser and Jean-Claude Latombe. Multi-modal motion planning in non-expansive spaces. *The International Journal of Robotics Reserach*, 29(7): 897-915, 2010.

[6] Marc Toussaint and Manuel Lopes. Multi-bound tree search for logic-geometric programming in co-operative manipulation domains. *In Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA 2017)*, 2017.

[7] Fabien Lagriffoul, Dimitar Dimitrov, Julien Bidot, Alessandro Saffiotti, and Lars Karlsson. Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research*, 2014. doi: 10.1177/0278364914545811.

[8] Igor Mordatch, Emanuel Todorov, and Zoran Popvic. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 31(4):43. 2012.

[9] Tomás Lozano-Pérez and Leslie Pack Kaelbling. A constraint-based method for solving sequential manipulation planning problems. *In Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference On*, pages 3684-3691. IEEE, 2014.

[10] L.P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. *In Robotics and Automation (ICRA), 2011 IEEE International Conference On*, pages 1470-1477, 2011.

[11] Siddharth Srivastava, Eugene Fang, Loreanzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through and extensible planner-independent interface layer. *In Robotics and Automation (IRCA), 2014 IEEE International Conference On*, pages 639-646. IEEE, 2014.