Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2017-007 January 30, 2017

Planning Robust Strategies for Constructing Multi-object Arrangements

Ariel Anders, Leslie Kaelbling, and Tomas Lozano-Perez

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Planning Robust Strategies for Constructing Multi-object Arrangements

Ariel Anders, Leslie Kaelbling, and Tomas Lozano-Perez

***Abstract*—A crucial challenge in robotics is achieving reliable results in spite of sensing and control uncertainty. In this work, we explore the conformant planning approach to robot manipulation. In particular, we tackle the problem of pushing multiple objects simultaneously to achieve a specified arrangement without external sensing. Conformant planning is a belief-state planning problem. A belief state is the set of all possible states of the world, and the goal is to find a sequence of actions that will bring an initial belief state to a goal belief state. To do forward belief-state planning, we created a deterministic belief-state transition model from supervised learning based on off-line physics simulations. We demonstrate the success of this approach in simulations and physical robot experiments.**

I. Introduction

Assembly of multi-part structures is a crucial application for robots and remains a significant challenge. The fundamental goal in an assembly problem is to achieve a set of relative position constraints among a set of parts; in general, these constraints require contact. These contacts are difficult or impossible to perceive visually and difficult or impossible to achieve via open-loop positioning actions.

A key strategy for robustly constructing assemblies is to use actions, such as pushing and compliant motions, that achieve desired contact relationships between objects by *exploiting the task mechanics*. For example, the robot can achieve contact between two object faces by pushing one up against the other, or achieve an insertion by using a remote-center compliance strategy. These actions have a tendency to act as "funnels," [16, 14] which can map a large set of possible initial configurations into a more compact set.

In general, the result of these actions is non-deterministic, depending on the actual (unobserved) initial state of the world and (unobserved) properties of the robot and objects. Given this setting, we are faced with the question of how to plan sequences of actions that can achieve a desired goal state without the ability to sense the precise outcome of each action. We frame this problem as one of *conformant planning*: given a set of possible initial object configurations, a set of actions with non-deterministic outcomes, a set of possible initial object configurations, and a set of goal configurations, the objective is to find a sequence of actions that is guaranteed to drive the objects into a configuration satisfying the goal.

The problem of conformant planning can be seen as a search through a space of *belief states*, which are sets of possible configurations of the objects in the assembly. Each primitive action is modeled using a transition function that maps an initial belief state into a resulting belief state; the resulting belief state is the union of the possible configurations resulting from applying that action primitive to each configuration in the initial belief state.

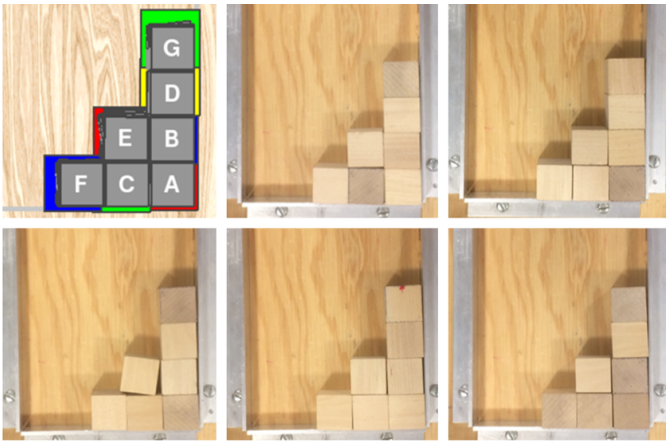


Fig. 1. Robust execution of 7 block assembly on a real robot; results shown for 1000 noisy simulated executions and 5 executions on a real robot.

For all but the simplest primitive actions in the most ideal circumstances, it is very difficult to directly model this transition function, which may be affected by detailed physical properties of the objects and robot in question. For these reasons, we use machine-learning methods to acquire belief-space transition models from simulations. Instead of attempting to learn action transitions for individual objects or sets of objects and aggregating these to derive a belief-state transition function, we learn transitions for the belief state of individual objects, given local context. We then compose the overall belief state transition from those of the individual objects.

Our focus in this paper is developing an approach to learning belief-space transition models. Although we demonstrate the learned model in the context of conformant planning, with no external sensing, it could also be used to implement a belief-space replanning strategy [21] that would incorporate intermediate sensing. The conformant setting offers a simpler, cleaner framework for evaluating the effectiveness of the model.

In this paper, we begin by describing a very general class of conformant assembly planning problems and outlining a generic search-based solution strategy for them. We then describe a particular instance of this problem class, in which the robot pushes objects resting on a table into a specified configuration, as shown in Figure 1. This task requires interaction between the robot and the pushed objects and also among

multiple objects. We present methods for learning the compositional transition model from a physics-based simulation. We finish by describing experiments and on both simulated and physical robots, which demonstrate effective planning and execution for assemblies of up to nine objects.

Related work A key challenge in the "exploit task mechanics" approach to manipulation is to understand the task mechanics well enough to be able to choose actions to achieve the desired outcomes. A number of operations, notably pushing, have been the subject of extensive study, and analytic models have been derived for the task mechanics; Mason [17] provides a helpful overview. In many cases, however, the outcomes of actions depend on physical properties that we do not have access to, such as pressure distributions or frictional coefficients. Nevertheless, the understanding of task mechanics can in some cases be exploited to plan robust manipulation. For example, Lynch and Mason [15] showed how to exploit multiple contacts to produce stable pushing trajectories and Dogar and Srinivasa [3] showed how to reliably funnel a range of initial locations of an object with known shape into a target location in the hand.

Recently, there have been a number of examples of "physics-based manipulation," which exploit the availability of physics simulation engines, originally developed for computer games, to predict the effect of actions in situations where analytic methods would be cumbersome at best. Most relevant to our work is the work on rearrangement planning [8], which tackles pushing an object to a target location in the presence of clutter, objects that are allowed to be pushed out of the way. Planning these operations relies on being able to predict, using physics simulations, the motion of multiple interacting objects being pushed by the robot. Related work on grasping through clutter also relies on such simulations [2, 9]. Note that this work is generally focused on placing or grasping a single-object, even though several objects (clutter) will need to be moved in the process; it does not address the goal of achieving a final arrangement of multiple objects.

An alternative to using analytical models or on-line simulation for physics-based planning is to learn compact models from experience (real or simulated). In the observable case, where the state of the world after an action can be determined, this is a simple regression problem: given many observations of *(state, action, next state)* learn a function to predict the *next state* given *(state, action)*. Given such a learned function, planning proceeds as before. A number of instances of this approach exist [22, 4, 18]. It is also possible to try to learn a policy directly and bypass planning; for example, Laskey et al. [12] learn a policy for grasping in clutter. However, such learned policies tend not to generalize as well as learning a model and then planning.

The work described above typically assumes that the initial state is known and the actions are reliable. In the presence of uncertainty in the initial placements of objects or in the outcome of the actions, we are interested in finding actions, such as pushing and compliant motions, that reliably achieve the goal state in spite of this uncertainty and without assuming the availability of additional observations. This class of problems is known as *conformant planning*; it has been explored in robotics for planning compliant motions [14] and sequences of tray tilting operations [5], and, more recently, for rearrangement planning [10, 7].

More broadly, the notion of conformant planning has been explored in the AI community, starting with Kushmerick et al. [11] who addressed it within the framework of probabilistic planning and Goldman and Boddy [6] who used expressions in a logic of knowledge to characterize belief states. Early work from theoretical computer science [19] shows that finding a finite-horizon optimal policy for a completely unobservable MDP is NP-complete, making this class of problems more efficient to solve than POMDPs in general. Yu et al. [23] find that conformant planning is an effective strategy for a multi-robot "tag" domain.

In this paper we address a problem that is related to the physics-based rearrangement planning problem of [8, 10] but is substantially different; in particular, it requires assembling a number of objects into a specified pattern. A sequence of distinct trajectories, some placing, some pushing, must be planned to achieve the goal. Our approach involves learning a transition model for planning, but we learn a model for transitions in the belief-space, rather than for transitions in the underlying state space.

II. Conformant assembly planning

A conformant assembly planning problem is specified by $(\Omega, \mathcal{C}_F, A_\Omega, A_m, \mathcal{T}, G)$, where

- Ω is a set of n known rigid objects;
- $\mathcal{C}_F$ is a set of collision-free complete configurations of all n objects, a subset of the whole configuration space $\mathcal{C}$; we additionally define $\mathcal{C}_O$, where $O \subseteq \Omega$, to be the space of collision-free configurations of the subset of objects O (assuming the objects in $\Omega \setminus O$ are not present).
- for each $o \in \Omega$, A_o is a set of actions that introduce object o into the assembly; we let $A_\Omega = \cup_{o \in \Omega} A_o$;
- A_m is a set of actions that manipulate (change the configurations) of some or all objects currently in the assembly;
- for each $o \in \Omega$, each $a \in A_o$, and each subset of existing objects $O \subseteq \Omega \setminus \{o\}$, $\tau_{a,o,O} : \mathcal{C}_O \rightarrow \mathcal{P}(\mathcal{C}_{O \cup \{o\}})$ is a transition function characterizing the addition of object o to the assembly currently consisting of objects in O by mapping an initial configuration in $\mathcal{C}_O$ to a set of configurations that is a subset of $\mathcal{C}_{O \cup \{o\}}$ ($\mathcal{P}$ denotes the *powerset* operator); for each $a \in A_m$ and $O \subseteq \Omega$, $\tau_{a,O} : \mathcal{C}_O \rightarrow \mathcal{P}(\mathcal{C}_O)$ is a transition function characterizing the effects of manipulation action a on objects in the current assembly; let $\mathcal{T}$ be the union of these $\tau_{a,o,O}$ and $\tau_{a,O}$;
- $G \subset \mathcal{C}_F$ is the set of configurations that are successful assemblies.

The actions in A_Ω add a new object to the assembly (or, for example, place it on a surface near other elements of the assembly). The actions in A_m can be any sort of feedback strategy (for example, position, force, or impedance

controllers) that control the robot for some period of time and are guaranteed to terminate. In the most general case, these actions may affect the configurations of all of the objects in the assembly.

Our approach is to convert this problem into a forward search in *belief space*; a belief space (or information space [13]) is a space of elements that characterize the robot's information or uncertainty about its domain. Elements in a belief space are typically subsets of, or probability distributions over, the underlying world state space. In this work, we will let the belief space $\mathcal{B} = \mathcal{P}(\mathcal{C}_F)$, that is, the set of all subsets of $\mathcal{C}_F$, and $\mathcal{B}_O = \mathcal{P}(C_O)$ be the restriction to subset of objects O. The methods described here could be easily modified to work with a distributional definition of belief, but we restrict the discussion to subsets for clarity.

We have defined the transition models τ as mappings from a single configuration to a set of configurations; in some cases it may be more convenient to specify $\tau' : \mathcal{B} \rightarrow \mathcal{B}$, mapping a set of configurations into the set of possible resulting configurations; such a model can be directly constructed from τ as:

$$\tau'(b) = \bigcup_{c \in b} \tau(c) \quad .$$

In a continuous configuration space, it will be impossible to finitely represent all possible subsets of $\mathcal{C}_F$, and so given any concrete domain we will have to make a choice about how to represent elements of $\mathcal{B}$ and therefore which subset will be usable during the search. Let $\widehat{\mathcal{B}}$ be a set of compactly representable elements of $\mathcal{B}$; then we can define a transition model $\widehat{\tau} : \widehat{\mathcal{B}} \rightarrow \widehat{\mathcal{B}}$, such that $\widehat{\tau}(b)$ is a smallest element $\hat{b} \in \widehat{\mathcal{B}}$ such that $b \subseteq \hat{b}$. This is a conservative approximation to the original τ' in that it generates belief states that are correct (they contain all possible true configurations) but may not be as small as possible.

The size of the configuration space grows exponentially with the number of objects in the assembly, and the size of the belief space grows exponentially with the size of the configuration space (in the discrete case). In order to fight this curse of dimensionality it is generally necessary to use a factored representation of belief states. If we think of a configuration $c = \langle c_1, \ldots, c_n \rangle$, where $c_i \in \mathcal{C}_i$ is the configuration of object i, then $\mathcal{C} = \mathcal{C}_1 \times \ldots \times \mathcal{C}_n$ is the Cartesian product of the configuration spaces of the individual objects. We cannot generally represent $\mathcal{C}_F$ as a Cartesian product, however, due to collisions between the objects.

This decomposition leads us to the idea of representing the belief space as the product of independent beliefs about the configuration of each individual object, so that $\widehat{\mathcal{B}} = \widehat{\mathcal{B}}_1 \times \ldots \times \widehat{\mathcal{B}}_n$, where $\widehat{\mathcal{B}}_i \subset \mathcal{P}(\mathcal{C}_i)$. Representing sets of possible configurations of each object rather than sets of possible complete configurations is much more compact, although it is not as expressive. For example, it is necessary to augment such a factored representation with a constraint that the entire configuration must be collision free, so we will in general define

$$\widehat{\mathcal{B}} = (\widehat{\mathcal{B}}_1 \times \ldots \times \widehat{\mathcal{B}}_n) \cap \mathcal{C}_F \quad .$$

This structuring also offers opportunities for compact representation of the transition functions; in most cases, there is a strong principle of locality, so that only some dimensions of the belief are changed by taking an action. Given a belief state $\langle \hat{b}_1, \ldots, \hat{b}_n \rangle$ and an action $\tau_{a,O}$, there will only be a set $O^* \subseteq O$ of objects that could possibly be affected by the action, so

$$\tau_{a,O}(\hat{b}) = \langle f(\hat{b}_1), \ldots, f(\hat{b}_n) \rangle$$

where $f(\hat{b}_i) = \tau_{a,i}(\hat{b}_i)$ if $i \in O^*$ and $f(\hat{b}_i) = \hat{b}_i$ otherwise.

The fewer objects that are potentially affected by any given action, the simpler the transition model is to apply. A transition model may be highly local when many of the objects are not yet placed near the objects being manipulated or when objects are already largely in a rigid formation (against one another or fixed objects in the world) so they do not move when they are contacted by other objects. It will generally be difficult to hand-specify the transition models sufficiently accurately; in our example implementation we learned the $\widehat{\tau}_{a,i}$ models from simulated data.

Given a conformant assembly planning problem specification $(\Omega, \mathcal{C}_F, A_\Omega, A_m, \mathcal{T}, G)$ and a belief-state representation $\widehat{\mathcal{B}}$, we can straightforwardly use the A^* (or other forward search) algorithm to solve it, if we restrict the action sets A_Ω and A_m to be finite. There may be sample-based search strategies that are effective in infinite action spaces but we do not consider them here. The search problem given to A^* is:

- Initial state: the belief state containing a single element, which is the empty configuration $\{\langle\,\rangle\}$;
- Successor function:

$$\begin{aligned} \text{succ}(\hat{b}) = &\{\widehat{\tau}_{a,O}(\hat{b}) \mid a \in A_m\} \cup \\ &\{\widehat{\tau}_{a,o,O}(\hat{b}) \mid o \in \Omega \setminus O, a \in A_o\} \end{aligned}$$

  where $\hat{b}$ contains the set of objects O;
- Goal test: $g(\hat{b}) = \hat{b} \subseteq G$.

We illustrate the general class of conformant assembly planning problems with two related example domains, implemented in simulation and on a real robot, both of which use a manipulator arm to place and push objects into planar arrangements.

III. Assembly by pushing

In this section, we formalize two related concrete robotics problems as instances of conformant assembly planning. The goal is to create planar arrangements of objects in contact with one another and with fixed obstacles using a combination of "gross" and "fine" motions [14]. The gross motions use a PR2 robot hand to place objects near their target poses, but there is considerable error in object placement due to control and calibration error in the arm and to the objects often sticking slightly to the fingers when they are released. In addition, the robot's fingers are large and so objects cannot be placed directly next to other objects. The fine motions use the PR2 robot hand, holding a paddle, to push objects up against one another and the fixed obstacles, effectively aligning them and

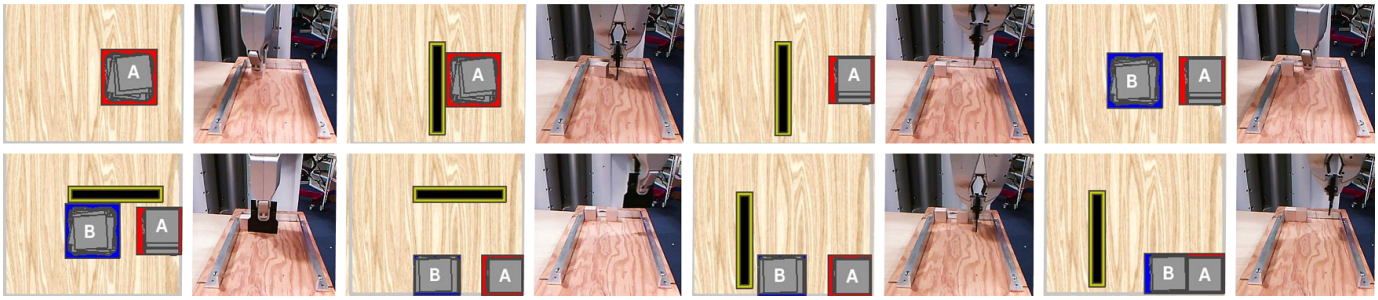


Fig. 2. Execution of a sample 8-step plan; in each pair, the left figure shows the results of 1000 simulations of the plan (gray blocks) as well as the predicted workspace bounding boxes from the transition model (in red and blue) and the right figure shows execution on the real robot.

moving them into place. In the first version of this problem, the fine motions only allow the paddle to contact a single-object; in the second version, we allow multiple objects to be pushed simultaneously with the paddle.

We can describe these problems as instances of the general conformant assembly planning problem as follows:

- The objects Ω are wooden cubes, 1 inch on a side, that can be placed on a planar surface, which has fixed obstacles in a "u" shape as shown in figure 1.
- The configuration space of each individual object $\mathcal{C}_i$ consists of its position and orientation in $\mathcal{SO}_2$, bounded by the workspace area; the free configuration space $\mathcal{C}_F$ of the whole system is the product of the $\mathcal{C}_i$ together with the constraint that the objects are not in collision with each other or the walls.
- The actions A_Ω add an object to the assembly by using a robot gripper to place an object into free space. Objects may be introduced into the assembly by placing them at a selected (x, y) position; the robot attempts to place the object so that it is rotationally aligned with the workspace, but there is significant error in the calibration, resulting in errors that are well bounded by $\pm 0.2\,\text{in}$ in x and y and $\pm 15^\circ$ in θ; in our current implementation we consider 9 possible values for (x, y) corresponding to constant offsets from the object's target position (x_g, y_g) in the goal.
- The actions A_m manipulate one or more objects already in the assembly through *push* actions, in which the robot holds a 2.5-inch paddle, places it at pose (x_p, y_p, θ_p) just above the surface of the table and attempts to move in the direction orthogonal to the face of the paddle; the controller has a low gain so that if it encounters obstacles during the motion they remain on the table and are pushed until they can move no further. In our implementation, the pushing angle θ is selected to be either 0 or $\pi/2$ (down or to the right) because the set of goals we consider is limited to arrangements of objects that are supported by the bottom and right walls. Our search strategy is currently limited to a discrete set of action choices, but it is important to choose these discrete values in a state-dependent way.
- The transition models τ are complex and will be the subject of section III-A.
- The goal test G is specified in terms of a workspace bounding-box $\langle x_g, y_g, \Delta x_g, \Delta y_g, \rangle$ for each object and is satisfied if each object can be guaranteed to be inside its bounding box.

The representable belief space for each object i, $\widehat{\mathcal{B}}_i$, is a "box" in $\mathcal{SO}_2$, represented with parameters $\langle x, y, \theta, \Delta x, \Delta y, \Delta\theta \rangle$ specifying the center and dimensions of the box; typically $\theta = 0$. We will often denote the center of a belief box by $q = \langle x, y, \theta \rangle$ and its dimensions (the "uncertainty") as $\Delta q = \langle \Delta x, \Delta y, \Delta\theta \rangle$. The representation of beliefs as boxes in the configuration space was chosen for compactness and computational efficiency, but it is clearly significantly restricted in the belief states it can represent. The complete belief space is the product of $\widehat{\mathcal{B}}_i$ for each object, together with the non-collision constraint.

We will frequently also make use of a conservative bounding box in the workspace $\langle x, y, \Delta_w x, \Delta_w y \rangle$, which can be computed from the configuration-space belief, where for a square block of dimension $2r$, we define

$$\text{BB}(\langle x, y, \theta, \Delta x, \Delta y, \Delta\theta \rangle) = \\ \langle x, y, r + \Delta x + r \sin(\theta + \Delta\theta), r + \Delta y + r \sin(\theta + \Delta\theta) \rangle$$

A belief state b satisfies the goal G if $\text{BB}(b)$ is entirely contained in $\langle x_g, y_g, \Delta x_g, \Delta y_g, \rangle$.

The parameters of possible push actions are selected depending on the current belief state. So, for a downward push, given a current belief state b, for each object i currently in the assembly, with workspace bounding box $\text{BB}(b_i) = (x, y, \Delta y, \Delta x)$, we let $\theta_p = 0$, $x_p = x + \Delta x + \epsilon$ and we consider several possible y_P values, corresponding to $y + \delta$ for $\delta \in \{-1.0, -0.5, 0.0, 0.5, 1.0\}$. For a rightward push, $\theta_p = \pi/2$, $x_p = x + \delta$, and $y_p = y + \delta y + \epsilon$.

The transition model for object placement is straightforward, simply appending the belief for the object being placed to the existing belief representation:

$$\tau_{(x,y,\theta),o,O}(\langle b_1, \ldots, b_m \rangle) = \langle b_1, \ldots, b_m, \langle x, y, \theta, d_x, d_y, d_\theta \rangle \rangle$$

where $m = |O|$ and (d_x, d_y, d_θ) are bounds on the error in the placement operation.

The transition model for push actions is significantly more complex, both because the results of pushing a single-object

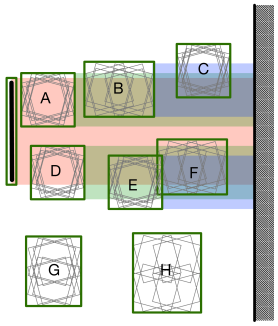


Fig. 3. Swept volumes and bounding boxes for computing contact graph.

are difficult to predict given variability in quantities such as surface texture and force applied by the robot, and because the robot may in general affect the state of several objects with a single push action. We discuss transition models for pushing in detail in the next section.

A. Structured transition model for pushing

The transition model for the push action has a structured decomposition and a local quantitative model that is learned from simulated data. Intuitively, the strategy is to find one or more sequences of blocks that will be pushed up against one another, constrained on one side by the robot paddle and on the other side by a wall (see Figure 2). Given such a sequence, we apply a learned quantitative local uncertainty model to compute the final center and delta values for the resulting object beliefs, starting from the object closest to the wall and working back toward the paddle.

Because we are ultimately going to use this model to search for a plan with reliable effects, it is only necessary to make predictions for actions for which we can confidently predict the posterior belief state. Thus, the transition model will be partial, in some situations declining to make a prediction. This partiality will allow us to maintain the correctness of the planner, but may cause it to be incomplete (in the sense that there may be problem instances for which a legal plan exists, but our system is unable to find it.)

The first step is to determine which objects are affected by the pushing operation. To do this, we construct a *contact graph* and extract *contact paths* of objects that are mutually constraining. The contact graph has a node R for the robot, a node W for the wall, and nodes for objects that are potentially moved by this operation. Figure 3 illustrates a belief state and the process of determining possible contacts. It contains 6 objects, the robot paddle on the left and a wall on the right. For each object, we draw 8 copies of the object's shape, each of which represents a vertex of the belief box in configuration space; the green box is the workspace bounding box, BB, associated with each object. Note that the bounding boxes for objects E and F overlap; this is allowed and makes sense if we recall the implicit non-collision constraint. Uncertainty in the robot's position and orientation are represented by a green workspace bounding box.

```
MAKECG((x, y, θ), b):
    agenda = {robot}; V = {robot}; E = { }; E' = { }
    while not EMPTY(agenda):
        o = agenda.pop()
        vol = sweptVol(BB(b[o]), (x, y, θ))
        for o' ∈ Ω ∪ {wall}:
            if BB(b[o']) ∩ vol ≠ ∅:
                if o' ∉ V: agenda.ADD(o')
                V.procadd(o')
                E.ADD((o, o'))
    return V, E, E'

PUSHTRANS(b, (x, y, θ)):
    V, E, E' = MAKECG((x, y, θ), b)
    P = MAXIMALPATHS(V, E ∪ E')
    b' = copy(b)
    maxPaddleX = None; maxBlocks = None
    for p ∈ SORTEDLONGESTFIRST(P):
        if maxPaddleX == None:
            maxPaddleX = wall.x − (len)(p) · d
            maxBlocks = len(p)
        if len(p) < maxBlocks:
            firstObjB = b[p[1]]
            if firstObjB.x − firstObjB.Δx > maxPaddleX:
                pass // Objects sure not to be moved
            else
                return None // Objects moved unpredictably
        if p ∩ E' ≠ ∅:
            return None
        for i = LEN(p) − 1 downto 1:
            b'[p[i]] = PUSHTRANSOBJ(p[i − 1], p[i], p[i + 1], b', θ)
    return b'

PUSHTRANSOBJ(b, prev, cur, next, θ):
    // Just handling case of push to right
    Δx', Δy', Δθ' = PREDICT(b[prev], b[cur], b[next])
    x' = b[next].x − d − Δx'/2
    return (x', b[cur].y, 0, Δx', Δy', Δθ')
```

Fig. 4. Pseudo code for computing the transition function on belief states for the push (x, y, θ) action.

The procedure MAKECG shown in figure 4 outlines the process formally. For simplicity, we just describe the case for a push toward the right; for other axis-aligned pushes it is essentially the same, up to changing signs and/or swapping x and y coordinates.

In the example of figure 3, execution would go as follows:

- We begin with the robot and compute the volume of the workspace it would sweep through if it was able to move all the way to the wall; this is shown in light red in the figure.
- We add each object whose bounding box overlaps this swept volume to the graph, and add an edge from the robot node to the object to the set E. In this case, we add edges to objects A, through F.
- Now, we take each of these objects in turn, computing their swept volume and adding arcs to objects they will contact.
- The swept volume for A is shown in green. It overlaps with objects B and C.
- The swept volume for B is shown in blue. It overlaps with object C.
- Similar processing is done on D, E, and F.

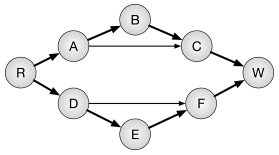


Fig. 5. Example contact graph; for clarity some arcs are omitted: nodes R and W are connected to all other nodes.

Figure 5 shows the resulting contact graph.

The next step is to find maximal contact paths between nodes R and W. A contact path p is maximal if there is no other path p' between R and W such that $p \subset p'$. These paths represent "trains" of objects that will end up nearly in contact with one another, pushed between the hand and the wall. In this example, there are two maximal paths: $RABCW$ and $RDEFW$. If the paths are not of the same length, then the prediction process also fails. Consider the case in which object D is not present in this example; the robot hand will compress objects A, B, and C against the wall but leave E and F unaffected. However, if D were present, but E and/or F were missing, then D would be pushed, but without a constraint on its right, we are unable to reliably predict its resulting position and uncertainty; in this case, the prediction model returns **None**, indicating that this operation cannot be used to construct a plan.

For each valid contact path, we use a local predictive model compositionally to compute the posterior belief for each object. The predictive model takes as input the belief states of three objects that occur sequentially in the contact path. In the *single-object* pushing model, we disallow any operations in which there is more than one path of influence between the robot and the wall.

B. Learning a quantitative model for pushing

We will work with a factored version of τ', which maps the belief state of an object and its neighbors on either side to its resulting belief state. In the transition model for a push to the right, we begin by predicting the uncertainties in the resulting object position, using a function PREDICT, which is learned from data. It has inputs and outputs in the form:

$$\Delta q' = \langle \Delta x', \Delta y', \Delta\theta' \rangle = \text{PREDICT}(b[prev], b[cur], b[next]) \ ,$$

where the inputs to the procedure are the current beliefs about three sequential objects in a pushing sequence. We assume that the center of the resulting uncertainty box has the same y coordinate as before and that the median rotation is 0. The median x coordinate is computed by finding the median x coordinate of the object to its right, subtracting the dimension of an object, d, and then subtracting the resulting x uncertainty, $\Delta x'/2$.

The problem of learning the PREDICT function can be treated as a supervised regression problem with three output dimensions. Again, for simplicity of exposition, we limit our attention to the push-right action. The inputs to the PREDICT procedure are $(q_p, \Delta q_p, q_c, \Delta q_c, q_n, \Delta q_n)$. We compute from these inputs a feature vector ϕ:

$$\langle \Delta q_p, \Delta q_c, \Delta q_n, q_p.y - q_c.y, q_n.y - q_c.y, ct \rangle$$

For a fixed object, such as a wall, which can only appear as the last object, we assume $\Delta q_n = (0, 0, 0)$, and that $q_n.y - q_c.y = 0$. The last feature in this vector, ct, encodes the types of the objects involved in this contact; it can take on the values in $\{row, roo, ooo, oow\}$ where r stands for $robot$, o for $object$ and w.

Because generating training data on the real robot would be prohibitively costly in time, we generate training data using a Box2D simulation [1]. We construct a data set of 1800 examples, using the following process to construct each example:

- An initial belief state b is randomly constructed, with Δx and Δy values drawn uniformly in the interval $[0.0, 0.4]$ inches and $\Delta\theta$ values drawn uniformly in the interval $[0, 15]^\circ$; b may contain 1, 2, or 3 blocks.
- For 1000 iterations, an initial state is drawn uniformly from b and constructed in Box2D, with the robot paddle offset to the left and a fixed wall, parallel to the paddle, to the right of all the objects. The paddle's y coordinate is randomly varied to generate a variety of offset values. The static and kinetic friction coefficients for the simulation of robot, table, and objects are drawn uniformly in the range $[0.25, 0.55]$. The robot's paddle is moved in the desired direction using a position controller to a distant set-point with gains of 1.0 for position and angular error; the controller is run for 50 simulation steps with $\Delta_t = 0.01\ s$. The final pose of the each object (x_i, y_i, θ_i), together with its contact type, is recorded.
- The resulting belief state dimensions are computed as:

  $$\langle xmax - \min_i x_i, \max_i y_i - \min_i y_i, \max_i \theta_i - \min_i \theta_i \rangle$$

  where $xmax$ is the maximum possible x coordinate for this object, if all the objects to its right were perfectly aligned and as far to the right as possible.

This data is then used to train a multi-output random-forest regressor using the Scikit-Learn toolkit [20] using hyper-parameters that were found using a grid search with 8-fold cross-validation. We used 90% of the data for training and hyper-parameter optimization and held out 10% for final evaluation, in which we found a root-mean-squared test error of 0.509 in Δx, 0.079 in Δy and 2.799 in $\Delta\theta$.

C. Search

In section II we described a generic forward-search problem that could be solved using A^* and in the previous section we specified the necessary state space and successor function for our example assembly-by-pushing domain. We assign a cost of 1 to every action.

In order to make the A^* search process tractable, we specify two additional components: a dominance-based pruning method and a search heuristic.

We begin by observing that some belief states are contained by others: given two belief states, b and b', we say that b *dominates* b' if and only if:

- b and b' are both defined on the same set of objects O;
- for all $o \in O$: the configuration space cube $b[o]$ is a subset of the configuration space cube $b'[o]$, that is $b[o] \subseteq b'[o]$.

Intuitively, if b dominates b', then b may be a more useful state in the search: the objects have overlapping possible states and less uncertainty in b than in b'. Therefore, we consider a pruning strategy in which, during the process of A^* search, whenever it reaches a state b' that is dominated by some state b that has already been visited, then b' is not added to the agenda. This strategy can result in computational improvements but it does risk pruning out correct solution paths.

In addition, we experimented with two heuristics. The first, which is admissible, effectively acts as a binary filter on states, assigning infinite cost to any state in which the objects in a horizontal or vertical contact path are not in the order specified by the goal. It is not possible, given the operations in this space, to reach the goal from such a state. This heuristic provides some useful search guidance but is not very strong. We define an additional heuristic that is inadmissible in general, but highly effective at improving the speed of the search without much reduction in the solution quality. We define:

$$H(b, g) = \sum_{o \in \Omega} H(\mathrm{BB}(b[o]), g[o])$$

where

$$H(bo, go) = \begin{cases} 2 & \text{if } bo == \textbf{None} \\ 2 & \text{if } bo.x \not\subseteq go.x \text{ and } bo.y \not\subseteq go.y \\ 1 & \text{if } bo.x \subseteq go.x \text{ and } bo.y \not\subseteq go.y \\ 1 & \text{if } bo.x \not\subseteq go.x \text{ and } bo.y \subseteq go.y \\ 0 & \text{otherwise} \end{cases}$$

This heuristic estimates that it will take one place action and one push action to add a new object to the assembly and one push per object dimension that is not currently contained within its goal interval. It is inadmissible because it is possible to push multiple blocks at once.

IV. Results

In this section we present quantitative results of our approach in the assembly-by-pushing domain.

Planner performance We tested the planner for six assemblies with (1, 2, 3, 4, 7 and 9 blocks). We varied the goal tolerances (the dimension of the goal regions) between ± 0.1 inches and ± 0.5 inches. The initial placement uncertainty was, unless stated otherwise, ± 0.2 inches in x and y and ± 15 degrees rotation. When a plan was obtained, we simulated it 1000 times. We found that whenever a plan was found, all the simulations satisfied the goal. However, for tight goal conditions, especially for larger assemblies, the search can exceed our limit of 5000 search nodes. Figure 6 shows the goal tolerances for which plans were found within the search limit; note that the vertical axis shows decreasing goal tolerance. The single-object-push setting requires longer plans and therefore fails to find plans for the larger assemblies within the search budget.

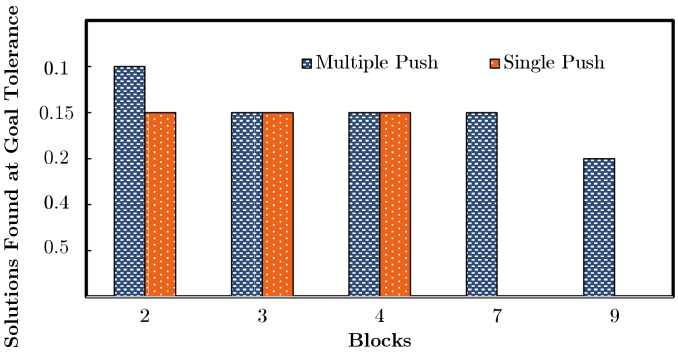


Fig. 6. Range of goal tolerances for plans found within 5000 node limit.

The search performance, as measured by number of search nodes expanded, is affected primarily by the quality of the heuristic used. In this experiment, the search node limit was 50,000. In the table in Figure 7, we see that using the "inadmissible" heuristic cuts the number of expanded nodes substantially, at the cost of longer plans. The domination test has a small beneficial effect given this heuristic, but a very large effect when a weaker or no heuristic is used.

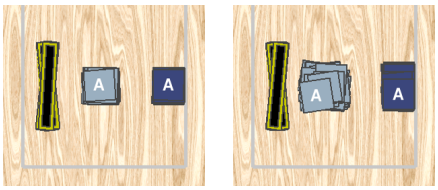
				1 Block		2 Blocks		3 Blocks		4 Blocks		7 Blocks	
Push Type	Search Type	Domination	Solutions Found	Nodes	Cost	Nodes	Cost	Nodes	Cost	Nodes	Cost	Nodes	Cost
Single	BFS	No	50.00%	8	3	186	6	17916	11	--	--	--	--
Single	BFS	Yes	50.00%	8	3	168	6	6457	11	--	--	--	--
Single	Admiss.	No	50.00%	12	3	153	6	4355	11	--	--	--	--
Single	Admiss.	Yes	50.00%	8	3	165	6	1766	11	--	--	--	--
Single	Inadmiss.	No	66.70%	6	3	14	6	71	11	82	15	--	--
Single	Inadmiss.	Yes	83.30%	6	3	13	6	129	12	73	16	10037	30
Multiple	BFS	No	50.00%	25	3	160	4	4483	6	--	--	--	--
Multiple	BFS	Yes	50.00%	27	3	191	4	2982	6	--	--	--	--
Multiple	Admiss.	No	50.00%	21	3	136	4	1149	6	--	--	--	--
Multiple	Admiss.	Yes	50.00%	27	3	179	4	1123	6	--	--	--	--
Multiple	Inadmiss.	No	100.00%	9	3	29	6	60	9	98	12	535	24
Multiple	Inadmiss.	Yes	83.30%	9	3	28	6	54	9	95	12	284	21

Fig. 7. Search Results Table.

The effect of uncertainty The learned transition function captures the "funnelling" behavior that is key to the success of the conformant plans. Figure 8 illustrates the reduction in uncertainty from pushing a block against the wall. Note that while the resulting uncertainty in x and θ are relatively insensitive to the magnitude of the initial uncertainty, the resulting uncertainty in the y direction is directly related to the initial uncertainty.

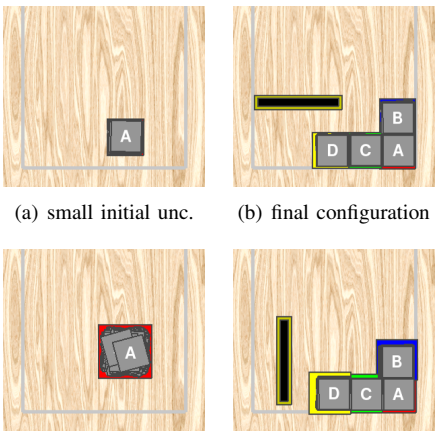
In Figure 9 we see that with small uncertainty in the place action (top row) and larger uncertainty (bottom row) we obtain comparable tight final placements; the gray blocks show results from 1000 simulations. Note that our learned model is conservative and predicts larger final uncertainty (the colored boxes) than was actually observed.

Figures 10 and 11 show the effects of goal tolerance and initial placement uncertainty on plan length. Tighter goals and higher initial uncertainty both increase the length of the required plans. Once the required plans exceed a length of



(a) (b)

Fig. 8. The effect of input uncertainty on output uncertainty: these images depict the results of 1000 physics simulations, where the initial location is drawn in the light colored box.



(a) small initial unc. (b) final configuration

(c) large initial unc. (d) final configuration

Fig. 9. The effect of initial placement uncertainty on the resulting assemblies.

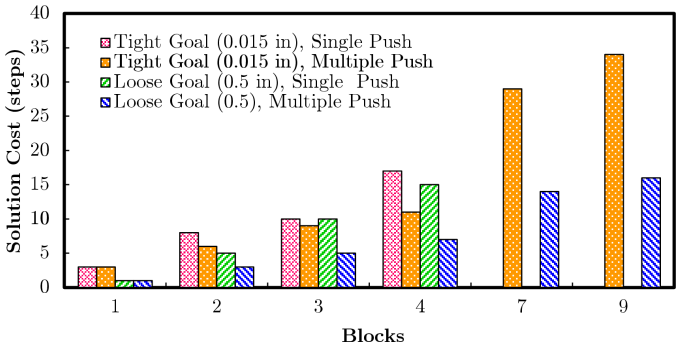


Fig. 10. The effect of multiple-push and goal tolerance on solution length. Tight goal is ± 0.15 in, loose goal is ± 0.5 in.

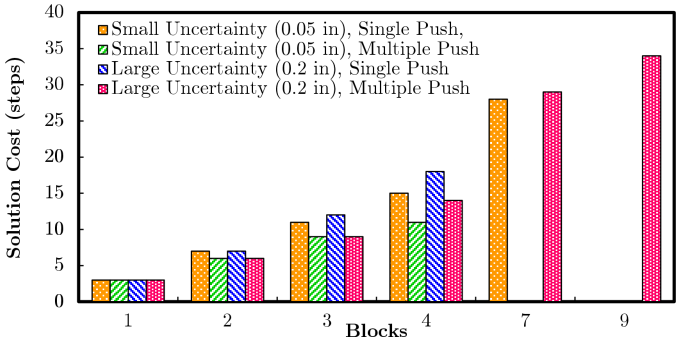


Fig. 11. The effect of multiple-push and initial place uncertainty on solution length. Small uncertainty is ± 0.05 in, large uncertainty is ± 0.2 in; angle uncertainty is ± 15 deg.

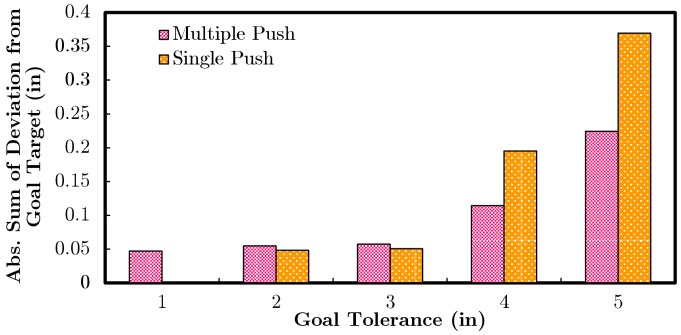


Fig. 12. The effect of varying goal tolerances on goal deviation.

around 16, the search process exceeds the allowed number of search nodes (5000); in such cases there is no corresponding result plotted in the graphs. Multi-push plans are generally shorter than single-push plans. Figure 12 shows that multi-push plans generally place the objects closer to their target locations, as measured by the sum of the absolute deviations in x and y. This is due to the fact that a multiple-object push tends to move more than one object towards its goal, often "patting" them into place.

Real robot experiments An example of a simple assembly sequence found by the planner, as well as its execution, can be seen in Figure 2; the accompanying video shows many more examples.

We obtained plans for 5 different assemblies (with 2, 3, 4, 7 and 9 blocks) and executed each one on the robot 5 times. The placement uncertainty was set at ± 0.2 inches and ± 15 degrees rotation; the goal tolerance was set to ± 0.2 inches. We saw one execution failure (in a 2-block assembly) during the 25 assemblies, for a 96% success rate.

Execution failures appear to be due to initial placements outside of the modelled placement uncertainty bounds. Increasing the modelled placement uncertainty could decrease this type of failure, at the expense of increasing the planning and execution times for the typical case. This is an unavoidable trade-off in conformant planning that could be ameliorated by moving to a belief-space replanning paradigm that adds some sensing, such as the final position of the paddle after a push.

V. Conclusion and Future Work

We have shown how a belief-space transition model can be acquired from off-line physics-based simulations and used to plan reliable planar push-assemblies in the presence of substantial uncertainty.

We believe that this approach can be extended to a variety of other settings, including the use of compliant motions instead of pushing and to assembling (simple) three-dimensional parts. We are also interested in exploring the use of the robot's other hand to create ad-hoc "jigs" to enable funneling actions. As we indicated above, it should be relatively easy to add observation actions and, under assumption of most-likely observations, implement a belief-space replanning approach along the lines of Platt et al. [21].

References

[1] Erin Catto. Box2D: A 2D physics engine for games. http://box2d.org, 2008. Accessed: 2017-01-30.

[2] M. Dogar, Kaijen Hsiao, Matei Ciocarlie, and Siddhartha Srinivasa. Physics-based grasp planning through clutter. *Robotics: Science and systems VIII*, 2012.

[3] Mehmet Dogar and Siddhartha Srinivasa. A framework for push-grasping in clutter. *Robotics: Science and systems VII*, 1, 2011.

[4] Sarah Elliott, Michelle Valente, and Maya Cakmak. Making objects graspable in confined environments through push and pull manipulation with a tool. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4851–4858. IEEE, 2016.

[5] Michael A. Erdmann and Matthew T. Mason. An exploration of sensorless manipulation. *IEEE Journal on Robotics and Automation*, 4(4):369–379, 1988.

[6] Robert P. Goldman and Mark S. Boddy. Expressive planning and explicit knowledge. In *International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, volume 96, pages 110–117, 1996.

[7] Aaron M. Johnson, Jennifer E. King, and Siddhartha Srinivasa. Convergent planning. *IEEE Robotics and Automation Letters*, 1(2):1044–1051, 2016.

[8] Jennifer E. King, Joshua A. Haustein, Siddhartha Srinivasa, and Tamim Asfour. Nonprehensile whole arm rearrangement planning on physics manifolds. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2508–2515. IEEE, 2015.

[9] Nikita Kitaev, Igor Mordatch, Sachin Patil, and Pieter Abbeel. Physics-based trajectory optimization for grasping in cluttered environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3102–3109. IEEE, 2015.

[10] Michael C. Koval, Jennifer E. King, Nancy S. Pollard, and Siddhartha Srinivasa. Robust trajectory selection for rearrangement planning as a multi-armed bandit problem. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2678–2685. IEEE/RSJ, 2015.

[11] Nicholas Kushmerick, Steve Hanks, and Daniel S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1):239–286, 1995.

[12] Michael Laskey, Jonathan Lee, Caleb Chuck, David Gealy, Wesley Hsieh, Florian T. Pokorny, Anca D. Dragan, and Ken Goldberg. Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations. In *IEEE Conference on Automation Science and Engineering (CASE)*. IEEE, 2016.

[13] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[14] Tomas Lozano-Perez, Matthew T. Mason, and Russell H. Taylor. Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, 3(1):3–24, 1984.

[15] Kevin M. Lynch and Matthew T. Mason. Stable pushing: Mechanics, controllability, and planning. *The International Journal of Robotics Research*, 15(6):533–556, 1996.

[16] Matthew Mason. The mechanics of manipulation. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 544–548. IEEE, 1985.

[17] Matthew T. Mason. *Mechanics of robotic manipulation*. MIT press, 2001.

[18] Tekin Meriçli, Manuela Veloso, and H. Levent Akın. Push-manipulation of complex passive mobile objects using experimentally acquired motion models. *Autonomous Robots*, 38(3):317–329, 2015.

[19] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Proceedings of the Robotics Science and Systems Conference*, 2010.

[22] Jonathan Scholz and Mike Stilman. Combining motion planning and optimization for flexible robot manipulation. In *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pages 80–85. IEEE, 2010.

[23] C. Yu, J. Chuang, B. Gerkey, G. Gordon, and AY Ng. Open loop plans in POMDPs. Technical report, Stanford University CS Dept, 2005.