Task-Oriented Planning for Manipulating Articulated Mechanisms under Model Uncertainty

Venkatraman Narayanan[†] and Maxim Likhachev[†]

***Abstract*— Personal robots need to manipulate a variety of articulated mechanisms as part of day-to-day tasks. These tasks are often specific, goal-driven, and permit very little bootstrap time for learning the articulation type. In this work, we address the problem of purposefully manipulating an articulated object, with uncertainty in the type of articulation. To this end, we provide two primary contributions: first, an efficient planning algorithm that, given a set of candidate articulation models, is able to correctly identify the underlying model and simultaneously complete a task; and second, a representation for articulated objects called the Generalized Kinematic Graph (GK-Graph), that allows for modeling complex mechanisms whose articulation varies as a function of the state space. Finally, we provide a practical method to auto-generate candidate articulation models from RGB-D data and present extensive results on the PR2 robot to demonstrate the utility of our representation and the efficiency of our planner.**

I. INTRODUCTION

A typical domestic task such as preparing a meal would require a robot to open cabinets in the kitchen, fetch plates from drawers, move items on the countertop, and open the sink tap. Many of these subtasks require the robot to understand the kinematic structure of the object being manipulated. Since these are commonly encountered tasks, it is natural to expect manufacturers of personal robots to preload them with a set of typical articulation models to accelerate planning. For instance, when the robot is required to open a door, it might already deduce from the instruction that the mechanism is a revolute joint, and only needs to reason if the door needs to be pushed or pulled, without having to learn the articulation from scratch.

Much of the recent work on articulated objects deals exclusively with learning the articulation type given observations of the object over time [7, 14], and taking actions that enable quick identification [2, 11]. While it would be useful to identify the complete articulation structure by interacting with the object, it would be inefficient for personal robots to fully characterize a model when a specific task needs to be accomplished. On the other hand, the planning process would be more efficient when bootstrapped with a set of candidate models.

This paper provides several contributions towards task-oriented manipulation of articulated objects under uncertainty in their model types. As a first contribution, we develop an efficient planner that can reason with uncertainty over a set of candidate models, to achieve a specific task. While the full planning problem with uncertainty in articulation model is an intractable POMDP (also noted in [2]), we show that we can reduce it to a manageable belief MDP under certain assumptions. This belief MDP can then be solved in real time using an MDP solver, making our approach practically attractive. The particular planner we use for solving the belief MDP is based on LAO* [4], an efficient heuristic search technique for solving MDPs. To briefly summarize, LAO* interleaves forward exploration of the state space using a heuristic, such as the one used by A* search [5], and dynamic programming backup updates, à la value iteration. The efficiency of the algorithm arises from carefully ordering the sequence of dynamic updates, and by ignoring parts of the state space that would never be reached from the start state under an optimal policy.

The second contribution of the paper is in extending the representation of articulated objects in order to allow for efficient planning under uncertainty in the mechanism type. Consider again the task of opening a door. While it is tempting to dismiss it as a simple revolute mechanism, it has its own complexities. More often than not, the door handle, which by itself is another revolute mechanism, controls the articulation of the door. With the door handle unturned, the door remains rigidly fixed, while turning the handle immediately activates the other revolute degree of freedom. To plan for manipulation of such objects, it is imperative that the representation used by the planner captures such complexities. The representation that we provide builds upon a commonly used representation in literature, the kinematic graph [14, 3]. The edges in the graph model the articulation between rigid components of an object, which are represented by the vertices of the graph. Consequently, it is sufficient to learn the parameters of the kinematic graph to understand the articulation of the object. We extend the kinematic graph to a Generalized Kinematic Graph (GK-Graph), which allows for state-dependent variation in the edge parameters of the graph. Using this representation, it now becomes possible to capture non-trivial mechanisms, such as the kinematic relation between the door and the wall, with dependency on the state of the handle.

Finally, even with a database of articulation models, it may be necessary to select relevant models for a particular task given only the sensor data. Further, once the candidate models are selected, they need to be converted into the representation used by the planner. For this, we present a practical method to auto-generate the candidate representations to be input to the planner, by extracting kinematically relevant visual features from sensor data, and then using a graph

This research was sponsored by ARL, under the Robotics CTA program grant W911NF-10-2-0016.

[†]Robotics Institute, Carnegie Mellon University, Pittsburgh, USA {venkatraman,maxim} at cs.cmu.edu

segmentation algorithm to build the GK-Graph.

In addition to the contributions discussed thus far, this work, to our knowledge, is the first to formulate the problem of manipulating unknown articulated objects as a planning under uncertainty problem. We demonstrate the end-to-end pipeline on the PR2 robot, a mobile manipulation platform, and provide experimental results that validate our claims.

II. RELATED WORK

Recently, there has been widespread interest in articulated objects in the robotics community. Katz et al. [7] identify the kinematic structure of planar articulated objects by tracking features on the object. They cluster feature trajectories from a moving object into rigid components and identify joint relationships between each pair by studying the rigid body transform of one with respect to the other. The same group extended their work to handle identification of 3D articulated objects in [8]. Sturm et al. [14] presented a comprehensive probabilistic framework for identifying the articulation of an object from an observed data sequence. They formulate the problem as finding the most likely kinematic graph [3] that explains the observations. This is done by maximizing the posterior likelihood over possible kinematic graphs, while using Bayesian Information Criterion to restrict model complexity of the fit.

In [9], Katz et al. use a relational representation for actions and formulate a relational Markov Decision Process (MDP) to compute a policy that would minimize the number of actions needed to discover the degrees of freedom of an articulated object. Barragán et al. [2] formulate the problem of identifying the articulation of an object as that of requiring the entropy in the belief over the possible model types to be less than a particular value. They use a Bayesian filter which can assimilate information from a variety of sensing sources to maintain the belief over the articulation model types and choose actions that reduce the entropy in the belief. Additionally, their formulation allows for modeling articulated objects which are not just serial mechanisms. On a similar note, Otte et al. [11] formulate the problem of identifying an object's degrees of freedom as that of minimizing entropy in the current belief about the degrees of freedom.

Our work is fundamentally different from the two camps of work described above in that we are interested in purposefully manipulating an articulated object for a specific task, and not in learning the kinematic graph structure, or minimizing the number of actions required to identify the mechanism type. The proposed formulation as a planning problem allows the robot to learn only as much as is required about the object's kinematic structure to complete a desired task, in contrast to existing learning formulations that ignore the task at hand. However, there are similarities in the representation we use for our planning problem with those used in literature. While Sturm et al. [14] use the kinematic graph as a graphical model to infer the articulation structure and parameters, we use an extension of it as a representation for our planner. In [2], a general formulation

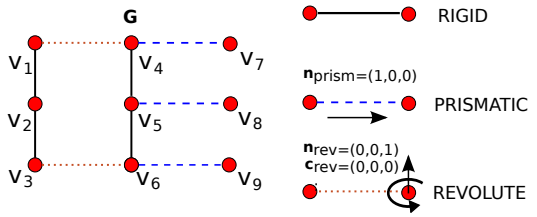


Fig. 1: Illustration depicting a GK-Graph containing 4 rigid edges (solid lines), 2 revolute edges (dotted lines) and 3 prismatic edges (dashed lines). Each vertex in the model represents the 6-DoF position and orientation of a local coordinate frame centered at that vertex. In the figure, n_{prism} is the prismatic axis direction, n_{rev} is the revolute axis direction and c_{rev} is a point on the revolute axis.

is used that allows for mechanisms to switch parameters over time. However, this generalization is not grounded in the perceptual capabilities of a robot. The Generalized Kinematic Graph that we use for our planner representation is an extension of the kinematic graph that provides the generalization described in [2], while at the same time, can be grounded in the perceptual capabilities—such as by using fiducial markers in the same vein as [14].

In another class of work, Jain and Kemp [6] propose the use of shared haptic data to identify specific instances of an object type while manipulating it. They demonstrate their method on the task of identifying a specific door instance, such as a cabinet door or a refrigerator door, by using a data-driven approach. The same authors, collaborating with Sturm et al. [15], present a feedback approach for learning the articulation model based on a robot's end-effector trajectory, and then predicting the trajectory based on the most likely model estimated by the learning framework. This approach is adopted in works by the same group [12] for operating doors and drawers. Again, our work differs from these methods as our goal is not identification of articulation models, but that of planning to complete a given task as quickly as possible under uncertainty over model types.

III. GENERALIZED KINEMATIC GRAPH

The Generalized Kinematic Graph (GK-Graph) is a directed graph that represents the kinematic relationship between features of interest or rigid components of an object. It is based on the kinematic graph [3], but extends it in that it allows for modeling mechanisms where the relationship between two rigid components can change as a function of their poses. Vertices v_i in the GK-Graph correspond to 6-DoF poses of features on the articulated object, and the edges parameterize the kinematic relationship between a pair of vertices. An edge e from v_i to v_j is a tuple $\langle m, \beta \rangle$, where m is the kinematic relation type (such as prismatic or revolute), and β is a vector of parameters needed to describe m. For example, if v_1 were a feature point on the wall and v_2 a feature point on the door, then the edge from v_1 to v_2 has $m =$ *revolute* and $\beta = \{\vec{n}_{\text{axis}}, \vec{c}_{\text{point}}, \theta_{\text{min}}, \theta_{\text{max}}\}$. All the parameters specified by β, in this case the axis of rotation, a point on the axis, and rotation limits, are in the local

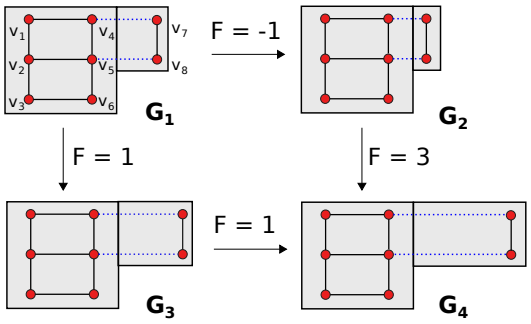


Fig. 2: GK-Graph states for a 1-D drawer. Solid lines in the GK-Graph represent rigid edges, while the dashed lines represent prismatic edges. The transition forces are exerted at vertex v_7, with positive forces corresponding to opening the drawer, and negative ones closing the drawer.

coordinate frame of v_1. Fig. 1 shows another example of a GK-Graph.

The GK-Graph differs from the kinematic graph because we allow the edge tuple $\langle m, \beta \rangle$ to be a function of the current coordinates of the vertices $f(V)$, where $V = \{v_1, v_2, \ldots, v_k\}$. This makes the GK-Graph a complete state for all mechanism models whose articulation varies only as a function of its configuration. That is, if we knew $f(V)$, then a future state of the object can be computed given the current one and the action applied. This addition increases the expressiveness of the GK-Graph, and allows it to capture complex articulated mechanisms. For example, the positions of the vertices that represent a door handle affect whether the relationship between the door and the wall is a rigid or revolute one.

Fig. 2 shows an example of how the GK-Graph represents a state for a 1-D drawer. GK-graph G_1 is the initial state of the object, with 8 vertices and 10 edges. The edges between $v_4 \rightarrow v_7$, and $v_5 \rightarrow v_8$ are prismatic, while all others are rigid. Applying a force of -1 on v_7 pushes the drawer in, and results in the state G_2, while a positive force of 1 opens the drawer, producing the state G_3. Applying a unit force on point v_7 in G_3, and a force of 3 on point v_7 in G_2 both result in the same configuration G_4. While simplistic, this example indicates how a GK-Graph might be useful for planning. Additionally, the actions that affect the GK-Graph state could be more sophisticated than simple forces applied on contact points on the object. The degree to which we can compute the resulting state depends on the fidelity of the simulation used and the input action types it supports. Since the focus of this work is on efficient planning, we will use a basic rigid body simulator that supports articulation constraints, while not worrying about such details as friction.

IV. PLANNING UNDER MODEL UNCERTAINTY

A. Problem Formulation

First, we assume that there are N candidate mechanism models, out of which one represents the true mechanism of the object. The planning problem then is to complete a specific task, whose goal is expressed as a particular configuration of the GK-Graph. This configuration could also be partially specified—for instance, the goal might be to have the GK-Graph vertex corresponding to the door handle to be at a particular 6-DoF pose in the workspace. We will use $x \in \mathcal{X}$ to denote the set of vertices in the GK-Graph. Each candidate mechanism model $\theta \in \{1, 2, \ldots, N\}$ defines a function $f_\theta(x)$ which specifies the edge tuples in the GK-Graph as a function of the vertices x. Let A denote the space of possible actions that can be applied on the object.

Since θ is unobserved, the planning problem as formulated is a POMDP with states $s \in \mathcal{S}$ denoting the tuple $\langle x, \theta \rangle$. Let $b \in \mathcal{B}$ be a belief state representing a probability distribution over s. Solving the problem as a POMDP would require us to compute the optimal policy $\pi^* : \mathcal{B} \rightarrow A$. A naive approach would be inefficient since the dimensionality of $\mathcal{B}$ is $N|\mathcal{X}|$, in addition to b being continuous.

We make two simplifying assumptions that reduce the problem to a tractable MDP on the belief state that can then be solved efficiently. First, we assume that the uncertainty lies only in θ, which is equivalent to saying that the vertices of the GK-Graph are fully observable—i.e, the set of 6-DoF poses on the object can be tracked without uncertainty. Second, we assume that the GK-graph transition model is deterministic. That is, given θ, the vertices x and action a, the next state x' is completely determined by $x' = \textsc{Sim}(x, \theta, a)$, where SIM is the simulator for the GK-Graph.

Since x is completely observed, the POMDP can be re-written as a Mixed-Observability MDP, or MOMDP [10]. Specifically, we only need to keep track of beliefs over θ for a given x. Let this be denoted as $b_x(\theta)$. The dimensionality of this new belief space is only N, the number of candidate mechanism models. Following [10], we can factorize the transition update for the MOMDP after executing action a at $b_x(\theta)$ as:

$$b'_{x'}(\theta') = \eta \sum_{\theta} p(x'|x, \theta, a) p(\theta'|x, \theta, a, x') b_x(\theta), \quad (1)$$

where η is a normalization constant. Because the transition model for s was assumed to be deterministic, and θ does not change over the planning horizon (i.e, θ has no motion model), the above expression simplifies to

$$\begin{aligned} b'_{x'}(\theta') &= \eta \sum_{\theta} \mathbb{1}_{x'}(\textsc{Sim}(x, \theta, a)) \mathbb{1}_{\theta'}(\theta) b_x(\theta) \\ &= \eta \mathbb{1}_{x'}(\textsc{Sim}(x, \theta', a)) b_x(\theta') \\ &= \begin{cases} \eta b_x(\theta') & \text{if } x' = \textsc{Sim}(x, \theta', a) \\ 0 & \text{otherwise} \end{cases} \end{aligned} \quad (2)$$

where $\mathbb{1}(\cdot)$ is the indicator function. A key result from the above is that the number of successors for a belief state $b_x(\theta)$, given an action a, is finite, and bounded by the number of mechanism models. This greatly simplifies the planning problem. Note that we do not include observations explicitly in updating the belief. This is because our only observation is x, the 6-DoF poses which are already part of the state.

B. Planning on the Belief MDP

Now that we have a tractable belief MDP (b-MDP) where the number of successors for a given belief state is

Algorithm 1 LAO* for b-MDPs

1: **procedure** COMPUTEPOLICY(b_{start})
2: $G \leftarrow \{b_{\text{start}}\}$
3: $G^* \leftarrow \{b_{\text{start}}\}$
4: **while** G^* has non-terminal states **do**
5: $T \leftarrow$ DFSTRAVERSAL(G^*)
6: **for all** $b \in T$ **do**
7: **if** b is not yet expanded **then**
8: $[S, P, C] \leftarrow$ GETSUCCESSORS(b)
9: $b' \leftarrow S_i$
10: **for all** $i \in \{1, 2, \ldots, |S|\}$ **do**
11: $v(b') \leftarrow$ GETHEURISTIC(b')
12: $G \leftarrow G \cup \{b'\}$
13: $a_{\text{best}}(b) \leftarrow \text{argmin}_{a \in A(b)} \big(\text{cost}(b, a) + \sum_{b' \in \text{SUCC}(b,a)} p(b'|b, a)v(b')\big)$
14: $v(s) \leftarrow \text{cost}(b, a_{\text{best}}(b)) + \sum_{b' \in \text{SUCC}(b, a_{\text{best}}(b))} p(b'|b, a)v(b')$
15: update the best partial solution graph G^* from G
16: IMPROVEVALUES(G^*)
17: **return** G^*

Algorithm 2 Successor Generation for b-MDP States

1: **procedure** GETSUCCESSORS(b)
2: $S = \{\}$ // successors
3: $P = \{\}$ // transition probabilities
4: $C = \{\}$ // costs
5: $x \leftarrow b.x$; $\bar{\theta} \leftarrow b.\bar{\theta}$
6: **for all** $a \in A(x)$ **do**
7: **for all** $i \in \{1, 2, \ldots, n\}$ **do**
8: $x'_i = \text{SIM}(x, \theta, a)$
9: **for all** i such that x_i is unique **do**
10: $\bar{\theta}'$ = zeros[N]
11: $p \leftarrow 0$
12: **for all** j such that $x_i = x_j$ **do**
13: $\bar{\theta}'[j] = \bar{\theta}[j]$
14: $p \leftarrow p + \bar{\theta}[j]$
15: $\bar{\theta}' \leftarrow$ NORMALIZE($\bar{\theta}'$)
16: $b' \leftarrow \langle x'_i, \bar{\theta}' \rangle$
17: $S \leftarrow S \cup \{b'\}$
18: $P \leftarrow P \cup \{p\}$
19: $C \leftarrow C \cup \{\text{cost}(x, a, x'_i)\}$
return $\{S, P, C\}$

Algorithm 3 Sense-Plan-Act Cycle

1: **procedure** MAIN()
2: **while not** SATISFIESGOAL(b_{start}) **do**
3: $\pi \leftarrow$ COMPUTEPOLICY(b_{start})
4: BEGINPOLICYEXECUTION(π)
5: wait for new observation z
6: $b_{\text{start}} \leftarrow$ UPDATEBELIEF($z, b_{\text{start}}, \pi_{\text{executed}}$)

bounded by the number of candidate models, we can use any MDP solver to obtain a policy over the b-MDP. For notational convenience, we will represent $b_x(\theta)$ as b and assume that b has two components: $b.x$, the 6-DoF poses and $b.\bar{\theta}$, an N-vector representing the probability distribution over θ. Given a start state $b_{start} = \langle x_{start}, \bar{\theta}_{start} \rangle$, the planning problem now is to find the optimal policy π^*_b : REACHABLE(b_{start}) $\rightarrow A$, where REACHABLE(b) is the set of all belief states that can be reached starting from b, under the optimal policy. Since the successors of a state are determined easily for a given belief state b, LAO* (shown in Algorithm 1) is an attractive algorithm for solving this problem as it efficiently interleaves forward expansions and dynamic backup updates while ensuring that we never look at the states unreachable from the start under an optimal policy [4]. LAO* maintains a partial solution graph G^* that stores all the belief states reachable from b_{start} under the current policy, and the corresponding best action to take at each belief state. Then, it iteratively performs the following two operations until there are no non-terminal states in G^*: first, a depth-first traversal of G^* is obtained, and for every state in the traversal order, it is expanded (Algorithm 2) if not a terminal state, and then a backup update is performed (lines 13 and 14 in Algorithm 1). The heuristic for a leaf state (line 11) is an admissible estimate of the cost-to-go for that state. Next, the best solution graph G^* is updated based on the newly computed v-values (cost-to-go), and the entire process is repeated again. Once there are no non-terminal states in G^*, value iteration is performed on the best solution graph (again in the DFS traversal order), until the v-values have converged to some ϵ-tolerance. The final solution graph G^* is returned as the policy to be executed.

C. Practical Considerations

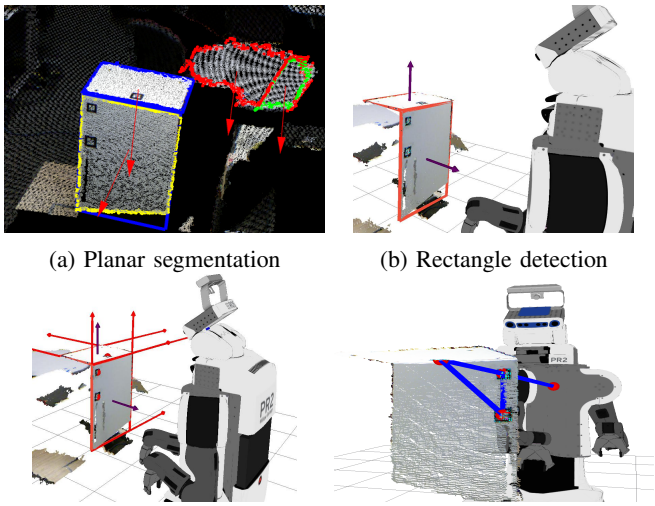
Although we compute a full policy by solving the belief MDP, we do so under the assumption that we will visit states only under the optimal policy. While this is fine in general, this would never be true in practice. There could be multiple sources of error, either from the simulation (the transition stochasticity) of the GK-Graph, or from the sensing, both of which were assumed to be non-stochastic in the planning phase. A practical approach to address this is to assume that stochasticity in observations received during execution arises in fact from the transition model, and then updating our belief before replanning again. From Eq. 1, we write the transition update for the belief

$$\begin{aligned} b'_z(\theta') &= \eta \sum_{\theta} p(z|x, \theta, a) p(\theta'|x, \theta, a, x') b_x(\theta) \\ &= \eta \sum_{\theta} p(z|x, \theta, a) \mathbb{1}_{\theta'}(\theta) b_x(\theta) \\ &= \eta p(z|x, \theta', a) b_x(\theta') \end{aligned} \tag{3}$$

where this time, we have not replaced $p(x'|x, \theta', a)$ with the indicator function. On the other hand, it would be reasonable to use a stochastic transition model here to ensure that the belief updates capture uncertainty to some extent. For instance, the particular belief update we use in our experiments is

$$b'_z(\theta') = \eta \mathcal{N}(z|\text{SIM}(x, \theta', a), \Sigma_{motion}) b_x(\theta') \tag{4}$$

where Σ_{motion} is a covariance matrix that captures the transition uncertainty. Once the belief update is done, we replan with the new start belief state $b'_z(\theta)$.



(a) Planar segmentation (b) Rectangle detection

(c) Generating candidate revolute and prismatic axes (d) Assigning edge tuples to the GK-Graph

Fig. 3: The perception pipeline for auto-generating candidate articulation models.

The main loop for replanning and updating the belief is shown in Algorithm 3.

V. EXPERIMENTS

We tested our planner on Willow Garage's PR2 robot for manipulating commonly encountered articulated objects. For these experiments, we used our perception system to auto-generate candidate mechanisms for each object, to provide an input to the planner. We first describe the details of our perception pipeline.

A. Perception Pipeline for Auto-generating Models

Since the focus of this work is on planning, we use fiducial markers on the objects to track the vertices of the GK-Graph. These fiducial markers are placed *randomly* on the object, with no attention to articulation details. The only requirement is that there is at least one pair of fiducial markers that can explain the underlying articulation. The tracked 6-DoF poses of the fiducial markers directly serve as the vertices of the GK-Graph. To compute the edges in the graph, we use a nearest neighbor method to connect every vertex with those within a radius of 50cm.

The next step is to generate candidate edge tuples for the GK-Graph. This is done as a two step process:

Estimating articulation axes: We use the Point Cloud Library [1] to segment out planes in the RGB-D sensor data. Next, for each plane segmentation, we run a RANSAC method for detecting rectangles in the contours of the plane. For each rectangle generated, we compute 5 articulation axes: 4 revolute axes, one for each edge of the rectangle, and 1 prismatic axis normal to the rectangle. These steps are shown in Fig. 3.

Mincut for GK-graph segmentation: Once the articulation axes are estimated, they need to be translated into edge tuples for the GK-Graph, given the arbitrary positions of the fiducial markers. This is done using a mincut operation on the GK-Graph [13], with edge weights computed as follows: Let x_i and x_j be the cartesian 3D locations of the vertices v_i and v_j. Let $d_{ij} = \|x_i - x_j\|$ and w_{ij} be the weight assigned for edge e_{ij} to compute the mincut. We define a model specific angle θ, which for prismatic models is the angle between $x_i - x_j$, the edge and the prismatic axis. For revolute models, θ is the angle between the lever arms of x_1 to the revolute axis, and x_2 to the revolute axis.

The weights for the prismatic and revolute cases are assigned as follows:

$$w_{ij} = \begin{cases} \exp(-(\alpha \cdot d_{ij} + \beta \cdot \cos(\theta))) & \text{if prismatic} \\ \exp(-(\alpha \cdot d_{ij} + \beta \cdot \sin(\theta))) & \text{if revolute} \end{cases}$$

where α and β are weight factors controlling the tradeoff between spatial and kinematic components. The edges in the computed mincut are then assigned the corresponding kinematic model parameters (obtained from the rectangle segmentation), and the rest are assigned rigid model parameters. Intuitively, the spatial component encourages edges between vertices close to each other to have rigid model assignments, while the kinematic component encourages edges on a plane surface to have prismatic assignments for the prismatic case, and edges on a plane surface to have rigid assignments for the revolute case.

B. PR2 Experiments

We tested our closed loop perception-planning system on Willow Garage's PR2 robot, for manipulating a variety of objects such as drawers and cabinets in offices, and shelves, drawers and a refrigerator in the kitchen. Fig. 4 shows some of the snapshots from these experiments. The GK-Graph was setup using fiducial markers on the object, and candidate kinematic mechanisms were auto-generated as discussed in Sec. V-A. Grasp locations on the objects were specified using a simple virtual fixture, as done in the recent Darpa Robotics Challenge. Once a grasp location is provided, a new vertex is automatically added to the GK-Graph and connected to the nearest existing vertex with rigid model parameters. The initial belief state of the robot was set as a uniform prior over the candidate models. Since the task was to open these objects, the goal state was defined as a GK-Graph state in which the end-effector of the robot (or the grasp vertex in the GK-Graph) had moved a particular distance d_{goal} away from its initial position. While this seems to be an atypical way of specifying the goal, it approximates well a real-life task goal such as 'open the object until an item of interest inside is visible or reachable'. Our action set consisted of a set of 20 forces sampled uniformly on a unit sphere. The heuristic used for LAO* encouraged states with grasp vertices far from the start state to be expanded first. Specifically, for a belief state b, the heuristic was computed as $h(b) = \min(0, d_{goal} - \|b.x[v_{grasp}] - b_{start}.x[v_{grasp}]\|)$. Finally, to execute an action returned by the policy, we use an inverse kinematics controller to move the end-effector of the robot from the grasp vertex of the current belief state, to that of the next.

Fig. 4: The PR2 robot opening a variety of articulated objects in an office space and a kitchen.

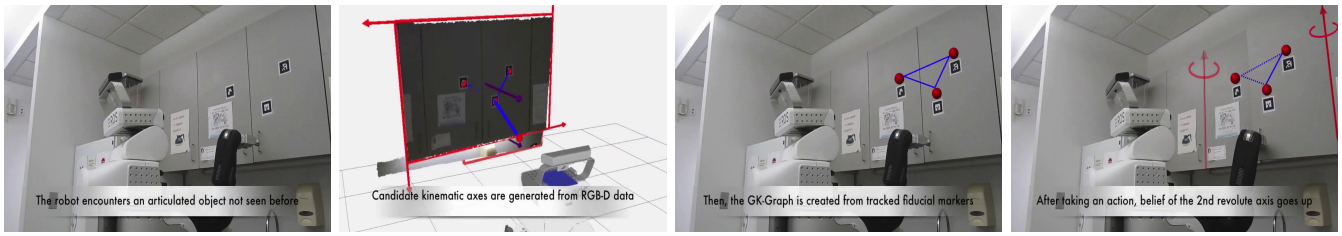


Fig. 5: Illustration of the end-to-end pipeline for manipulating articulated objects. The robot first encounters an unknown articulated object, a kitchen cabinet in this case. The perception system described in Sec. V-A generates candidate articulation models, which the planner then uses to generate a policy. After executing the first action, the robot observes the GK-Graph vertices (the fiducial markers) and updates its belief over the candidate models. In particular for this example, the robot believes that the revolute axis on the right is more likely than the one on the left after taking the first action. The planner again computes a new policy using this updated belief, and the process repeats until the task is completed

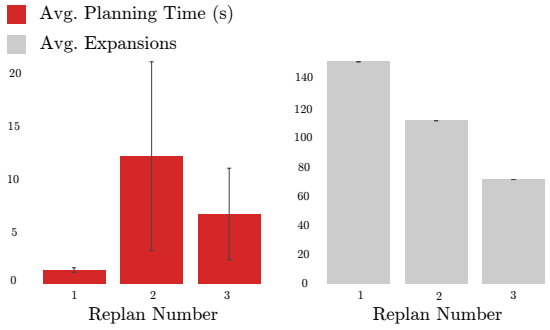
.

Using the approach described above, the PR2 robot could open every object shown in Fig. 4. While the system was successful in most of the trials, we observed two common failure modes. First, the planning does not include the kinematics of the robot manipulator. This leads to partial plans (cartesian trajectory of 6-DoF end-effector poses) that are infeasible for execution, thereby causing a standstill. The correct solution to this would be to plan in the combined belief space and configuration space of the robot. However, this increases the dimensionality of the problem dramatically and is perhaps unnecessarily complicated. Instead, a practical solution would be to obtain the cartesian end-effector trajectory in the usual way, and then use a full-body planner for the robot (joint planning in arm and base motions) to find a plan for the robot that is feasible for execution. The second failure mode arises when the fiducial markers on the object go out of the field of view, or are occluded by the robot during execution of a plan. Once again, while the correct solution would be to model the GK-Graph state as partially observable, a more practical solution would be to run an independent state estimator on the tracked GK-Graph.

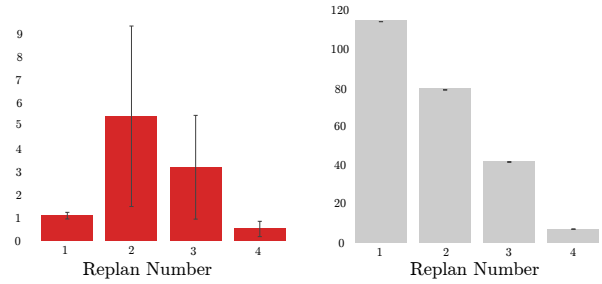
The associated video contains trials of the PR2 opening a variety of articulated objects, including those in Fig. 4. In Fig. 5, we show the end-to-end pipeline for manipulating an unknown articulated object.

C. Planner Efficiency Tests

To test the robustness of our method and quantify the planner's efficiency, we ran 25 trials each on an office cabinet and drawer, with the task being to open the object starting from a closed state. For the efficiency tests, the goals were specified as a particular end-effector locations that must be attained by the robot. The heuristic for LAO* is then simply the Euclidean distance between the 3D coordinates of the grasp vertex at the current state and the goal state. The success rate was 100% on these efficiency test trials. Fig. 6 shows the average planning time and expansions for the trials on opening drawers and cabinets, as a function of the replan number. In our trials, the robot had to replan up to 4 times during execution based on its observations. The statistics are computed over trials that had the same number of replans, and 'expansions' refers to the number of times the LAO* planner called for a simulation of a GK-Graph state. While the average number of expansions decreases monotonically with the replan number, the planning time does not. This is a consequence of the fact that although the underlying GK-Graph state may be the same, the time taken for value



(a) Statistics for opening drawers



(b) Statistics for opening cabinets

Fig. 6: Planning statistics for trials on opening drawers and cabinets.

iteration to converge on the partial solution graph depends on the initial belief state over the models. Since there is stochasticity in our observations over the trials, it is expected that the time taken for the v-values to converge will differ in each trial. This can also be seen from the error bars on the planning times.

VI. CONCLUSIONS

As a primary contribution, we have presented an efficient planner for manipulating articulated objects under uncertainty in the true mechanism of the object. The efficiency arises from assuming a deterministic transition model for the object given the true mechanism type, and a completely observable object state, which then reduce the original intractable POMDP planning problem to a belief MDP with a finite number of successor states. By using a heuristic MDP solver (LAO*) that discards extraneous parts of the belief space, we can run a real-time planning and execution loop. We also provide an expressive representation of articulated objects, namely the Generalized Kinematic Graph, for modeling complex kinematic mechanisms, and a heuristic method based on graph segmentation for auto-generating candidate articulation models from RGB-D data. Extensive trials and demonstrations on the PR2 robot showcase the capability of our planner, and its real-time applicability in personal robot manipulation tasks.

A natural extension and future work is to handle stochastic transition models for the object and sensing uncertainty. While this would be expected to improve planner performance and success rates, it is unclear if there is structure in the problem that can be exploited for solving the complete POMDP. Also, an interesting direction for future work would be to handle candidate models that are only approximate or under-specified. This now introduces a transition model for the unobserved part of the state. Although this would increase the planning complexity, there is still structure in the problem since the object state is completely observed.

REFERENCES

[1] Aitor Aldoma, Zoltan-Csaba Marton, Federico Tombari, Walter Wohlkinger, Christian Potthast, Bernhard Zeisl, Radu Bogdan Rusu, Suat Gedikli, and Markus Vincze. Point cloud library. *IEEE Robotics & Automation Magazine*, 1070(9932/12), 2012.

[2] Patrick R Barragán, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Interactive bayesian identification of kinematic mechanisms.

[3] Roy Featherstone and David Orin. Dynamics. siciliano, bruno and khatib, oussama (eds.), handbook of robotics. 2008.

[4] Eric A Hansen and Shlomo Zilberstein. Lao: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1):35–62, 2001.

[5] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[6] Advait Jain and Charles C Kemp. Improving robot manipulation with data-driven object-centric models of everyday forces. *Autonomous Robots*, 35(2-3):143–159, 2013.

[7] Dov Katz and Oliver Brock. Manipulating articulated objects with interactive perception. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 272–277. IEEE, 2008.

[8] Dov Katz, Moslem Kazemi, J Andrew Bagnell, and Anthony Stentz. Interactive segmentation, tracking, and kinematic modeling of unknown 3d articulated objects. 2013.

[9] Dov Katz, Yuri Pyuro, and Oliver Brock. Learning to manipulate articulated objects in unstructured environments using a grounded relational representation. In *In Robotics: Science and Systems*. Citeseer, 2008.

[10] Sylvie CW Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, 29(8):1053–1068, 2010.

[11] Stefan Otte, Johannes Kulick, Marc Toussaint, and Oliver Brock. Entropy-based strategies for physical exploration of the environments degrees of freedom.

[12] Thomas Ruhr, Jürgen Sturm, Dejan Pangercic, Michael Beetz, and Daniel Cremers. A generalized framework for opening doors and drawers in kitchen environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3852–3858. IEEE, 2012.

[13] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44(4):585–591, 1997.

[14] Jürgen Sturm. Learning kinematic models of articulated objects. In *Approaches to Probabilistic Model Learning for Mobile Manipulation Robots*, pages 65–111. Springer, 2013.

[15] Jürgen Sturm, Advait Jain, Cyrill Stachniss, Charles C Kemp, and Wolfram Burgard. Operating articulated objects based on experience. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2739–2744. IEEE, 2010.