Using a compliant, unactuated tail to manipulate objects

Young-Ho Kim and Dylan A. Shell

***Abstract*—We demonstrate manipulation of objects using the dynamics of a rope-like structure attached to a mobile robot as a passive tail. Three challenges arise in modeling and planning: the physics involved is non-trivial, the tail is underactuated, and motions of the object are non-deterministic. For such systems, some actions are well-characterized by a simplified motion model (e.g., for dragging objects), but we resort to data-driven methods for others (e.g., striking motions). A sampling-based motion planner, adapted to deal with non-deterministic object motions, is used to optimize motion sequences based on a specified preference over a set of objectives, such as execution time, navigation cost, or collision likelihood. Experiments show that a robot with a passive tail can manipulate cylindrical objects with (quasi-static) dragging, dynamic striking motions, and combinations thereof. The method produces solutions that suit diverse preferences effectively, and we analyze the complementary nature of dynamic and quasi-static motions, showing that there exist regimes where transitions between the two are indeed desirable, as reflected in the plans produced.**

***Index Terms*—Manipulation Planning; Motion and Path Planning; Underactuated Robots; Flexible robots**

I. Introduction

PEOPLE use ropes and rope-like apparatus (*e.g.,* chains, cords, or lassos) for manipulation in a variety of qualitatively different ways. When wrapped around objects, flexible structures enable their user to exploit constrictional, tensional, and frictional forces to restrain the object being manipulated, sometimes also helping control the object by gripping it statically. In martial-arts cinema portrayals, whipping actions are most commonly used for reaching and attacking enemies (*e.g.,* Indiana Jones' famous bull whip, cowboys with their lassos, and Spiderman). Those whip-like actions, in contrast to winding and tying actions, are produced with flexible cords by exploiting the dynamics of the continuous, compliant structure. This paper investigates how a robot may use flexible rope-like structures as tools in diverse ways, demonstrating high-speed dynamic (*e.g.,* striking) and high-precision quasi-static (*e.g.,* dragging) actions, as well as admixtures thereof.

We study a novel robot system with a flexible rope-like structure attached as a tail (see Fig. 1). Three main challenges arise in using this system for manipulation. First, the interplay of the object, the robot, and its passive tail involves mutual influences that warrant non-trivial physics to describe their interactions, complicating the modeling of joint states (the object–robot–tail triple) and making their full description daunting. Secondly, the precision with which the tail's configuration can be controlled is limited because it is underactuated. Thirdly, the object's state transitions are governed by the tail, which imposes a level of indirection —the tail itself being mediated by the robot's motion— that means the transitions are non-deterministic. This paper details an approach that addresses these three challenges.

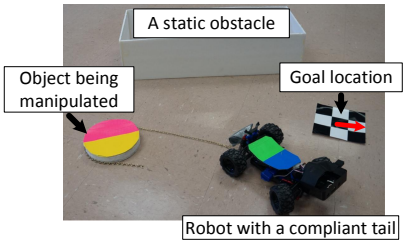


Fig. 1. A representative scenario: our goal is to manipulate the object from the initial pose to the goal pose. A novel idea is that the robot only employs its tail to translate and rotate the object.

We began by constructing a set of primitives that, while parsimonious, possesses sufficient richness to enable useful manipulation. Associated with each primitive motion is a forward model that is used to predict the object's subsequent position and orientation within the workspace. Finally, a sequence of primitive motion actions are sought to solve a given manipulation problem instance. We employ a sampling-based motion planning technique coupled with a particle-based representation to find such sequences.

The experiments we report constitute a unique demonstration of manipulation through the use of an unactuated tail, providing proof of sufficiency, and showing that though our approach involves gross simplifications, it nevertheless enables planning and successful execution of actions to manipulate objects within the workspace. The work also provides a basis for exploring the importance of several notions of path cost, complex preferences between objectives, and how a diverse portfolio of primitives enables synergy.

II. Related Work

Much closely related research involves modeling of and planning for flexible structures in tethered mobile robots or continuum manipulators. Absent from such work is consideration of high-speed motions that exploit the dynamics of the flexible component, the primary feature of this paper.

Prior efforts addressing manipulation of objects without special purpose effectors, includes work on pushing [1], [2],

Manuscript received: March 1, 2016; Revised: May 25, 2016; Accepted: June 26, 2016.

This paper was recommended for publication by Editor Han Ding upon evaluation of the Associate Editor and Reviewers' comments. This work was supported in part by the National Science Foundation as part of Grant IIS-1302393 and IIS-1453652.

Young-Ho Kim and Dylan A. Shell are with the Department of Computer Science and Engineering at Texas A&M University, College Station, Texas, USA. {yhkim, dshell}@cse.tamu.edu

Digital Object Identifier (DOI): see top of this page.

caging [3], or striking [4]. First Donald et al. [5], and Bhattacharya et al. [6] more recently, used a system of two robots connected by a rope to manipulate multiple differently shaped objects, capitalizing on the ability of the rope to adapt to the geometry of the objects and environment. Both efforts control the rope motions kinematically, without consideration of the masses of the ropes, the forces that lead to the motions, or the tension induced by accelerations.

Elephants' trunks and snakes have inspired several continuum mechanisms (theoretically hyper-redundant, but underactuated in practice [7], [8]). The research focuses on the design and control of such mechanisms with kinematic and dynamic models [8], [9]. Cowan and Walker [10] suggested possible dynamic motions for actively controlled continuum robots, including a flicking action. We are less concerned with specific models of tails and flexible devices than with dynamic actions and their potential for interaction with the environment. The method introduced here allows both dynamic and quasi-static actions; the dynamic actions are shown to be efficient motions for some objectives (*e.g.,* object distance per action).

For manipulation, it is non-trivial (or infeasible) to build analytical models that capture the full complexity of the object-tool-environment interactions involved. The inevitable consequence is uncertainty which must be dealt with in some way. Dogar and Srinivasa [11] reduced the uncertainty of a pushing action by utilizing the funneling effect of pushing. Related ideas include that of Meriçli et al. [2] who proposed an experience-based approach that uses past motions, and Phillips et al. [12] developed an online motion planning approach that makes use of information from previous searches. Berenson et al. [13] who demonstrated repair of paths from the past-learned paths using the Rapidly-exploring Random Tree (RRT). We extend the RRT framework using manipulation primitives and, similar to [14], we propagate motion errors using a particle-based representation.

Tails have been used in limited ways, primarily to enhance a robot's agility (stabilization/maneuverability [15]–[18], turning speed [19], [20], and dynamic response to disturbances [21]–[23]). All these prior works use active tails.

III. Preliminaries and Problem Description

Let M_o, M_r, and M_t represent the mass of the object, the robot, and the tail, respectively. We consider three coefficients of friction μ_o, μ_r, and μ_t for the object, robot, and tail, in their contact with the workspace floor. One additional coefficient of friction, μ_{o-t} is that between the object and tail. For a tail of length L, the physical system parameters form tuple $p = \langle M_o, \mu_o, M_r, \mu_r, M_t, \mu_t, L, \mu_{o-t} \rangle$. The object's state $X_o = (x, y, \theta) \in \chi$, has (x, y) as the position of the center of the object in $\mathbb{R}^2$ and θ its orientation. The robot's state X_r is defined analogously. We let $\tilde{X}_t$ be a vector modeling the tail configuration, either as an approximation via n segments, or a parametric function. Then the manipulation system can be treated as a transition function F. Using control $u \in U$, F makes the transition from $X(k)$ to another state $X(k+1)$:

$$X(k+1) = F(X(k), u(k); p), \tag{1}$$

where X contains states of X_o, X_r, and $\tilde{X}_t$.

A. Simplifying assumptions

Our robot has an inelastic tail that it uses for manipulating a cylindrical object, and the robot is assumed to be powerful enough to drag the tail along with the load of the object. Also, we treat the robot and the object states as observable, while the tail configurations are not.

B. Problem definition

Let $\mathcal{J}(X) = (J_1(X), \ldots, J_N(X))$ be a vector-valued cost function, where the integer $N \geq 2$ is the number of objectives that describe aspects of the system's performance. For instance, J_i could correspond to a measure of path safety, a measure of path accuracy, execution time, or navigation cost. **Problem:** Given U and p, manipulate the object from the initial pose $X_S = (x_s, y_s, \theta_s)$ to the target pose $X_G = (x_g, y_g, \theta_g)$, making all state transitions via the robot tail, finding a sequence of motions $\pi^* = [u_1^*, \ldots, u_N^*]$ that minimize $\mathcal{J}(X)$, constructed as a linear combination of the individual objectives

$$\mathcal{J}(X, \vec{\alpha}) = \sum_{i=1}^{k} \alpha_i \cdot J_i(X), \tag{2}$$

for some given $\vec{\alpha} = \alpha_{1,\ldots,k} \in \mathbb{R}^+$.

The total cost function for a trajectory $[X_0, \ldots, X_N]$ is

$$\mathcal{J}_{([X_0,\ldots,X_N],\vec{\alpha})} = \sum_{i=0}^{N} \mathcal{J}(X_i, \vec{\alpha}). \tag{3}$$

So the problem is to find

$$\pi^* = \underset{(X_0,\ldots,X_N)\in\chi}{\operatorname{argmin}} \mathcal{J}_{([X_0,\ldots,X_N],\vec{\alpha})}, \tag{4}$$

where $\vec{\alpha}$ expresses preferences over objectives, and N is the number of waypoints.

The overall system architecture appears in Fig 2.

IV. Modeling Motion Primitives

To simplify both the modeling and planning problems, we use a small set of motion primitives: each represents a simple and stable activity which is, ideally, helpful in achieving a goal. We desire a small portfolio of motion primitives, each contributing some aspect lacking in the others. Additionally, important factor is each primitive's amenability to modeling and, ultimately, its predictability; this is a function of primitive complexity. Three motion primitives, broadly representative of three broader classes, are considered: 1) quasi-static motion primitive — modeled as deterministic, 2) quasi static motion primitive involving stick-slip frictional transitions — modeled as non-deterministic, and 3) dynamics-based motion primitive where the tail strikes the object — which is non-deterministic.

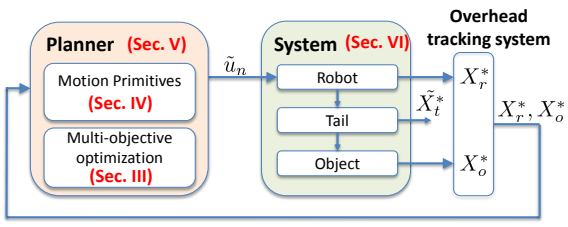


Fig. 2. An overview of the system showing sections with discussion.

From 1) to 3), the tractability and accuracy of modeling decrease owing to the inherent complexity of the physical phenomena involved. Our philosophy is to profit from diversification: we resort to a data-driven approach when the system is highly non-linear and transitions are too complicated for us to treat otherwise.

We consider a set of motions, each of which represents a sequence of controls, u, executed over an interval of time, which results in a transition between two states in the continuous state space. Each of these is characterized offline, either through analytical models, simulation, or calibration experiments (as will be described below). We also allow primitives to be parametrized, exploiting regularity and symmetries in so doing. Let $\tilde{u}(\phi)$ denote a set of motions, where we have expressed the fact that they are parameterized by angle ϕ, representing the direction in which to move the object. Then we define a motion primitive $\tilde{\mathcal{U}}(\phi)$ to be a tuple $\langle \tilde{u}^-(\phi), \tilde{u}(\phi), \tilde{u}^+(\phi) \rangle$, where $\tilde{u}^-$ represents an initialization motion, $\tilde{u}$ represents a motion moving the object, and $\tilde{u}^+$ represents a termination motion. The following sections discuss these in further detail.

Additionally, we made two design decisions:

***Design Decision** 1:* We do not represent the tail explicitly.

Rationale: This vastly simplifies the planning and representation problems. Our approach is to have each motion primitive $\tilde{\mathcal{U}}$ include an 'initialization' motion $\tilde{u}^-$, which normalizes the tail configuration, laying the tail out into some predictable configuration no matter its prior condition. This allows $\tilde{\mathcal{U}}$ to follow any state. In practice $\tilde{\mathcal{U}}$ is imperfect so the uncertainty of the tail configuration is implicitly included in the parameterized motion primitives, as discussed later.

***Design Decision** 2:* We assume the robot motion is deterministic, whereas we must treat the object motion as non-deterministic for any $\tilde{\mathcal{U}}$.

Rationale: The robot state and the object state are coupled via the tail. It is possible to estimate the robot state relative to the object state if the robot's control policy seeks to maximize this form of information. For example, the robot can stop the instant the object stops moving. Then, given $\tilde{\mathcal{U}}$, the object state can be used to help determine the relative location of the robot, a step which helps reduce planning complexity. There are small errors in pose estimates in practice, but are incorporated in the motion primitive model easily.

These two design decisions yield a simplification of Eq. (1) to give a transition function F_s as

$$X_o(k+1) = F_s(X_o(k), \tilde{\mathcal{U}}(k), p), \tag{5}$$

wherein we only need consider the object's state, because the robot state is determined by the object state.

A. Robot motion model

The robot has motion constraints that affect the feasible tail configurations and, ultimately, the object motions. Here we assume a simple car model for the robot. We generate the robot trajectories via Dubins curves that allow the shortest path in obstacle environments [24], ensuring a collision-free path for the robot. A Dubins curve consists of circular curves and linear motions, and all robot motions are treated as such, including atomic actions in each $\tilde{\mathcal{U}}$.

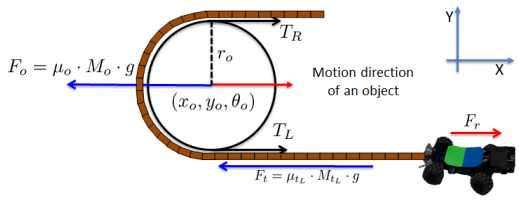


Fig. 3. A free body diagram for the dragging motion.

B. Quasi-static model: simplified analytic model for dragging motions

First, consider a dragging motion where the tail and object make contact throughout: The robot approaches the object's side, wraps the tail around the object, and moves forward, resulting in simultaneous rotation and translation of the object. There is no need for explicit consideration of the tail's configuration other than basic physical properties such as its mass and coefficients of friction. The following analysis considers the quasi-static regime, assuming small inertial forces compared to the tail's contact forces.

A free body diagram for this scenario is shown in Fig. 3. Suppose that the direction of the object's translation is in the horizontal x-direction (i.e., for $\phi = 0$). The basic dragging motion depends on tensions T_L and T_R, on the left and right sides of the tail, where the object is taken as the dividing point. We denote the mass of the left hand side of the tail by M_{t_L}, and the force induced by the robot by F_r. While dragging, F_r may increase as M_{t_L} increases, causing an increase in the left side of the tail's static friction F_t. Still, $F_r - F_t$ is constant when the robot's force F_r increases (e.g., via a feedback controller) to ensure the quasi-static motion is maintained. Eq. (6) and Eq.(7) are applications of Newton's second law, here for translation in the x-direction. When the right side of the tail is not moving due to friction with the floor, the operation is identical to the physics of an ideal pulley, permitting us to write $T_L = T_R$ in Eq. (7). The object exerts the total frictional force of $T_L + T_R = 2T$, meaning that if the robot moves distance of $2d$, the object moves distance of d.

$$T_L = F_r - F_t, \tag{6}$$
$$T_L + T_R = F_o. \tag{7}$$

Ideal pulley physics is only applicable when friction fixes the right side of the tail. As the robot drags its tail, the proportion of mass on the right (left) side decreases (increases, respectively). Eventually T_R cannot be sustained; the friction breaks down when the mass on the tail's right side is inadequate, which is dependant on the length of the tail. Let the minimum length of the right side of the tail be L_{min}, then there are two distinct frictional phases: 1) the right tail stays while the left tail moves; 2) the whole tail moves, slipping over the floor. For the first phase, we can get a closed-form solution as it is identical with a pulley physics system.

Focusing on this first phase of motion, we call this our fine

motion primitive, $\tilde{\mathcal{U}}_0$,

$$\begin{aligned} X_{o(x,y)}(k+1) &= d\,A + X_{o(x,y)}(k), \\ X_{o(\theta)}(k+1) &= X_{o(\theta)}(k) + \frac{2d}{r}, \\ X_{r(x,y)}(k+1) &= 2d\,A + X_{r(x,y)}(k), \end{aligned} \tag{8}$$

where d is a limited distance, depending on L, since the robot can move distance at most $L - L_{min}$. As before, ϕ denotes the direction of motion, but r is the object radius, and A is $[\cos\phi, \sin\phi]$.

The uncertainty in this motion primitive has its origins in the robot's motions. The robot executing $\tilde{\mathcal{U}}_0$ must ensure that it operates in the first frictional phase only. This requires that it stop during the dragging motion and then separate the tail from the object. This additional step (a termination step, $\tilde{u}_0^+$) is fairly complicated, but *is the price demanded for a primitive that realizes such a simple model.*

A second more cavalier primitive has the robot simply continue forward even after the tail and object begin to slip (in Fig. 3, it keeps going in the positive x-direction). Passing through the first frictional phase into the second, a highly non-linear transition from sticking to slipping occurs manifesting itself as additional uncertainty for this primitive $\tilde{\mathcal{U}}_1$.

The second dragging primitive, $\tilde{\mathcal{U}}_1$, has a pair of stochastic state transition functions:

$$\begin{aligned} X_o(k+1) &= X_o(k) + \omega_1, \quad \omega_1 \sim \mathcal{N}(X_{mean_1}, \Sigma_1), \\ X_r(k+1) &= X_o(k+1) + \omega_2, \quad \omega_2 \sim \mathcal{N}(X_L, \Sigma_2), \end{aligned} \tag{9}$$

where X_{mean_1} and Σ_1 are the mean and variance of the object state transitions via $\tilde{\mathcal{U}}_1$. Here X_L and Σ_2 describe the robot's state error relative to $X_o(k+1)$, parameterized by ϕ. The robot is dragging the object initially, but at some point the tail begins to slip, and eventually friction causes the object to come to rest. Our controller ensures that the robot stops then too. Thus, the robot's state distribution depends on the object location, which facilitates planning because the prediction of the robot state can be determined relative location of the object (and ϕ, and the tail length).

C. Dynamic model: high-speed striking

In addition, we investigated a high-speed primitive motion that has the robot exploiting the dynamics of the tail to lash the object. The robot approaches the object from an angle dependent on ϕ, then drives with constant velocity and at the maximum steering angle while keeping a short separation distance. As the tail is moved at high speed, its internal tension stiffens the rope along its extent, and when the object is struck both rotation and translation result.

The striking primitive, $\tilde{\mathcal{U}}_2$, has state transition function with a probability distribution:

$$\begin{aligned} X_o(k+1) &= X_o(k) + \omega_3, \quad \omega_3 \sim \mathcal{N}(X_{mean_3}, \Sigma_3), \\ X_r(k+1) &= X_o(k+1) + \omega_4, \quad \omega_4 \sim \mathcal{N}(X_{any}, \Sigma_4), \end{aligned} \tag{10}$$

where X_{mean_3} and Σ_3 are the mean and variance of the object state transitions via $\tilde{\mathcal{U}}_2$. The mean and variance of the relative robot state is given by X_{any} and Σ_4, which capture dependency on ϕ, and intrinsic robot control errors.

Specific values of X_{mean_1}, X_{mean_3}, Σ_1, Σ_2, Σ_3, and Σ_4 are reported in Appendix A.

D. Recap of motion primitives: initialization

The motion primitives require initialization motions, $\tilde{u}^-$, to ensure the tail is in a well-defined configuration. Some motion primitives also need additional actions, $\tilde{u}^+$, to simplify the robot's state estimation. This section reviews the motion primitives, detailing these additional motions which have not been described yet. We summarize the set of primitives we use in Fig. 4: from left to right tractability and accuracy of modeling decrease because the physical phenomena involved are complex. Data-driven models are used when the system is highly non-linear and transitions are too complicated to treat otherwise. Though not shown, primitive $\tilde{\mathcal{U}}_2$, has merit in terms of the object travel distance per unit of time.

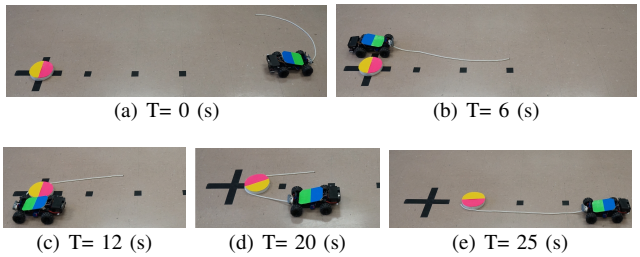
The photos in Fig. 5 show $\tilde{\mathcal{U}}_0$ and $\tilde{\mathcal{U}}_1$. There are four phases. The first sets the robot and object some distance (and bearing ϕ) apart—shown in Fig. 5(a). Then, the robot executes a pre-planned path, $\tilde{u}^-$, bringing the robot next to the object, as in Fig. 5(b). Next, in Fig. 5(c), the robot makes a simple surrounding motion that wraps the object, whereafter the robot drags the tail for distance of $2\,d$. For $\tilde{\mathcal{U}}_0$, the robot will stop in Fig. 5(d). For $\tilde{\mathcal{U}}_1$, the robot will continue until the object stops moving in Fig. 5(e).

For $\tilde{\mathcal{U}}_0$, the robot reverses over its tail to separate the tail from the object after the object has stopped. These additional motions, $\tilde{u}_0^+$, involve execution of a pre-planned path. These are not needed for $\tilde{\mathcal{U}}_1$, thus $\tilde{u}_1^+ = \varnothing$.

Fig. 6 shows the three phases of $\tilde{\mathcal{U}}_2$. Initially, the robot navigates to the initialization location via $\tilde{u}_2^-$, shown in Fig. 6(a). Second, the robot executes its high-speed motion with steering at hard lock. The tail configuration is predictable as it essentially becomes a semi-rigid body—Fig. 6(b). Next the tail hits the object, shown in Fig. 6(c), and the object moves. Finally, to be consistent with the model, the robot moves to a location relative to the object's resting pose, which is shown in Fig 6(d). This last motion is $\tilde{u}_2^+$.

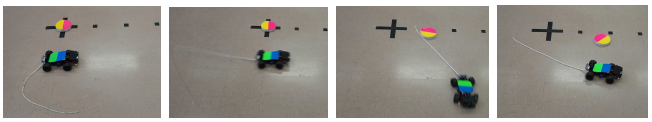
Motion Primitives	$\tilde{\mathcal{U}}_0$	$\tilde{\mathcal{U}}_1$	$\tilde{\mathcal{U}}_2$
Properties	Quasi-static idealized & interaction	Quasi-static involving stick/slip together	Dynamics-based
Model	Deterministic w/ small variance added	Stochastic	Data-driven (Stochastic)

Fig. 4. A diverse set of motion primitives are used. From left to right, the object travel distance per unit of time increases while the model tractability and model accuracy decrease.



(a) T= 0 (s) (b) T= 6 (s) (c) T= 12 (s) (d) T= 20 (s) (e) T= 25 (s)

Fig. 5. The initialization motion for $\tilde{u}_0^-$ and $\tilde{u}_1^-$ is from (a) to (b). Then, the robot drags its tail to move the object through (c), (d) and (e). If the robot wants to execute $\tilde{\mathcal{U}}_0$ only, the robot might stop execution as shown in (d). For $\tilde{\mathcal{U}}_1$, the robot keeps dragging until the object stops moving, shown in (e).



(a) T= 0 (s) (b) T= 1 (s) (c) T= 1.5 (s) (d) T= 2 (s)

Fig. 6. (a) The initialization,$\tilde{u}_2^-$, positions the robot relative to the object. (b) This shows a high-speed motion. The robot makes a circular motion. (c) The tail configuration is like a semi-rigid body after first round, and then the tail hits the object. (d) The robot stops at some location relative to the object ($\tilde{u}_2^+$).

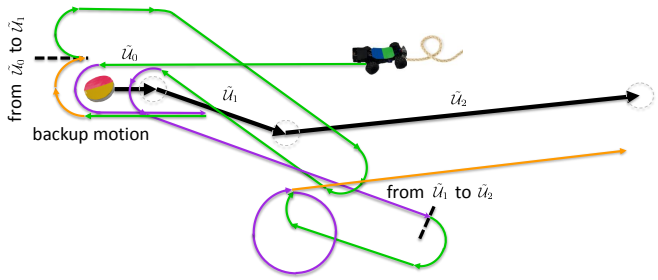


Fig. 7. Motion primitives are sequenced together. The black line shows segments of motion that the object undergoes. The green lines are the initialization motions, $\tilde{u}^-$, and the orange solid lines are termination motions, $\tilde{u}^+$. The purple lines are portions of the path which move the object. The broken black lines show points of transition between primitives.

V. Planning with the Motion Primitives

Having outlined the primitives individually, the next step is to plan and execute sequences of motion primitives. Fig. 7 provides an illustration: the path consists of the sequence $\tilde{\mathcal{U}}_0$, $\tilde{\mathcal{U}}_1$, and $\tilde{\mathcal{U}}_2$. First, the robot executes $\tilde{\mathcal{U}}_0$. The robot path is generated via a Dubins curve and the robot follows the green line as its initialization, $\tilde{u}_0^-$. It then makes the motion shown by the purple line to move the object. At some point, the robot stops, executes a backing up motion as $\tilde{u}_0^+$ to separate the tail from the object. Here a transition from $\tilde{\mathcal{U}}_0$ to $\tilde{\mathcal{U}}_1$ occurs. The robot moves along the (green) initialization path as $\tilde{u}_1^-$ and then the robot executes the dragging motion (purple). The final transition is from $\tilde{\mathcal{U}}_1$ to $\tilde{\mathcal{U}}_2$. The robot initializes with $\tilde{u}_2^-$ and then makes a high-speed circular motion. Once the object stops moving, the robot goes to its final location via $\tilde{u}_2^+$.

How does a planner find such a sequence, especially since the primitives include motion uncertainty? We use the Rapidly-exploring Random Tree (RRT) [25] and add a particle-based representation for uncertainty of the object's state transitions. The algorithm operates on a graph describing the object's state space. Each vertex has the object's current state, X_o and the robot current state X_r, and the belief states of each object state have a weighted set of N particles, $\{(X_{o_1}, w_1), \ldots, (X_{o_N}, w_N)\}$. Each edge is labeled with the motion primitive needed to traverse between the associated vertices for the object and the robot trajectories in Sec. IV-A.

Algorithm 1 RRT_PLANNING

1: INPUT: X_S, X_G, robot initial location X_{r_0}, initial tree $\tau_0(X_S)$
2: OUTPUT: $\pi^* = (\tilde{\mathcal{U}}_1^*, \ldots, \tilde{\mathcal{U}}_N^*)$
3: TREE_BUILDING_PHASE(X_S, X_{r_0}, $\tau_0(X_S)$)
4: SEARCH_PHASE(τ, $\vec{\alpha}$, X_G)

Algorithm 2 TREE_BUILDING_PHASE(X_S, X_{r_0}, $\tau_0(X_S)$,)

1: INPUT: $[\tilde{\mathcal{U}}_0, \ldots, \tilde{\mathcal{U}}_N]$, X_S, robot initial location X_{r_0}, an initial tree $\tau_0(X_S)$,
2: OUTPUT: τ_K.
3: **for** k = 1 to K **do**
4: $(X_{rand}, \tilde{\mathcal{U}}_{rand}(\phi))$ = SAMPLE()
5: X_{near} = NEAREST(X_{rand}, $\tilde{\mathcal{U}}_{rand}(\phi)$, τ_k)
6: **if** NEW_STATE(X_{rand}, $\tilde{\mathcal{U}}_{rand}(\phi)$, X_{near}, X_{new}) **then**
7: $X_{new}.particles$ = PROPAGATE(X_{near}, $\tilde{\mathcal{U}}_{rand}(\phi)$)
8: τ.add_vertex(X_{new}, $\tilde{\mathcal{U}}_{rand}(\phi)$)
9: τ.add_edge(X_{near}, X_{new}, $\tilde{\mathcal{U}}_{rand}(\phi)$)
10: **end if**
11: **end for**

Algorithm 3 $X_{new}.particles$ = PROPAGATE(X_{old}, $\tilde{\mathcal{U}}$, ϕ)

1: **for** p = 1 to P **do**
2: $X_{new}.particles_p$ = $X_{old}.particles_p$ + SAMPLE($\tilde{\mathcal{U}}, \phi$)
3: UPDATE_WEIGHT($X_{new}.particles_p$)
4: $X_{new}.particles_p.cost$ = UPDATE_COST($X_{new}.particles_p$)
5: **end for**

Alg. 1 gives the RRT algorithm with a mechanism adapted for propagation of errors. First, we build the tree with our motion primitives from the object start node. Then we search for a low-cost path based on the scalarized cost function. (In our implementation these procedures are performed offline.) Alg. 2 consists of following basic functions: The SAMPLE function returns uniform samples of the object state X_o in χ_{free}. We also sample a heading angle $\phi \in [0, 2\pi)$ and a specific motion primitive $\tilde{\mathcal{U}}$ with object rotation direction. The NEAREST function finds the nearest node X_{near} from τ_k. Then, the NEW_STATE function determines the new tree node based on global constraints, such as being collision-free (including the object state transitions and the robot trajectories). When new nodes are added, we propagate object motion errors. In Alg. 3, we compute a new particle state by sampling particles via the probability distribution of the motion primitive. Then we update the weights, normalizing them based on the average of $X_{new}.particles$, also updating $\mathcal{J}(X)$.

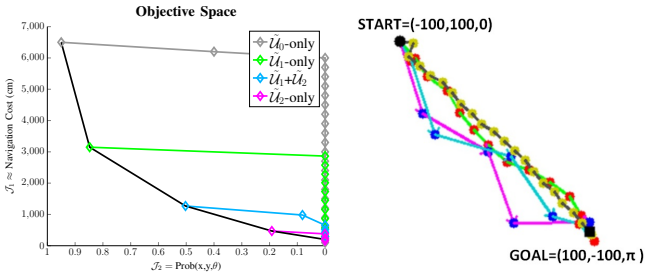
Once we have a random tree, the next step is to find a sequence of motion primitives via the SEARCH_PHASE in Alg. 1. We use Dijkstra's algorithm with $\mathcal{J}(X_i, \vec{\alpha})$.

VI. Experimental Results

This section discusses results from several physical robot experiments.

A. System setup

We used an RC car controllable to velocities between $0.4\ m/s$ for $\tilde{\mathcal{U}}_0$ and $\tilde{\mathcal{U}}_1$, and over $1.5\ m/s$ for $\tilde{\mathcal{U}}_2$. An embedded computer on the car controls the robot and communicates with a separate computer integrated with an overhead tracking system which localizes the object and the robot. The software uses the ROS framework. All experiments are conducted in our test arena of size $5.18\ m \times 4.27\ m$. Pose errors are estimated to be $\pm\ 5\ cm$ and $\pm\ 10rad$, respectively. The tail configuration was not tracked. The following physical properties were measured: the target object is a cylinder with $mass = 30\ g$, $\mu_o = 0.6655$, and $radius = 7.5\ cm$; the



(a) The Pareto front (in black); the x-axis is a measure of path accuracy; the y-axis is navigation cost.

(b) Four paths are plotted in 2-dimensional space (heading has been omitted).

Fig. 8. $\tilde{\mathcal{U}}_0$-*only* is a gray path. $\tilde{\mathcal{U}}_1$-*only* is a green path. $\tilde{\mathcal{U}}_2$-*only* is a magenta path. $\tilde{\mathcal{U}}_1$+$\tilde{\mathcal{U}}_2$ is a sky blue path.

robot weighs 700 g; tail is a chain with $mass = 35\ g$, $length = 70\ cm$, and $\mu_t = 0.5952$.

B. Scenarios: planners, environments, and objectives

1) Three planners: For comparison of the experimental results, we considered three types of planners. (1) The *simple* naïve planner in which the error propagated with a pre-planned fixed ϕ. This planner's anticipated uncertainty is greatly increased over the other two planners. (2) The *adjustable* planner updates the ϕ parameter based on the current object state. This update is in line 7 of Alg., 2. (3) The *adjustable + replanning* planner extends the previous planner by generating a new tree and a path to reduce any error that remains after the robot finishes each motion.

2) Two environments: We have two scenarios. The first is obstacle-free, in which we validate the approach extensively. The second environment has a static obstacle, and gives a clear demonstration of how different cost preferences (regarding collisions) are reflected in the resulting sequence of motion primitives.

3) Three objectives: We are interested in total execution time, a measure of path accuracy, and a measure of path safety, denoted J_1, J_2, and J_3, respectively. Two scaling degrees-of-freedom are suffice: α_1 and α_2, with $\alpha_3 = 1 - \alpha_1 - \alpha_2$. At transitions between motion primitives, we compute $\mathcal{J}(X)$ by accumulating each cost function. The execution time is the total robot navigation time. The total accuracy of the path includes both the reliability of object transitions between nodes, and how close the states are to the goal. They are computed via $1 - Pr(X_{oi})_{\hat{X}_{oi}}$ and $1 - Pr(X_{oi})_{X_G}$, where $\hat{X}$ is our model's prediction.

First, we examined the open space scenario to understand the basic characteristics of the motion transitions. For this scenario, we set $\alpha_3 = 0$ because there are no obstacles. Then, our four different preferences give four scaled parameters: the planner with $\alpha_1 = 0.00001$, $\tilde{\mathcal{U}}_0$-*only*, is the most conservative planner which gives a most reliable path albeit with most costly execution time. With $\alpha_1 = 0.0001$, $\tilde{\mathcal{U}}_1$-*only*, is modestly conservative, allowing a reliable path with high probability to go to the goal. The planner with $\alpha_1 = 0.0003$, $\tilde{\mathcal{U}}_2$-*only*, is most optimistic, allowing only high-speed motions. The planner with $\alpha_1 = 0.0002$, $\tilde{\mathcal{U}}_1 + \tilde{\mathcal{U}}_2$, is mixed.

As can be seen in Fig. 8, we have four kinds of paths: $\tilde{\mathcal{U}}_0$-*only* has sequences with twenty $\tilde{\mathcal{U}}_0$. $\tilde{\mathcal{U}}_1$-*only* has sequences with twelve $\tilde{\mathcal{U}}_1$. $\tilde{\mathcal{U}}_2$-*only* has sequences with four $\tilde{\mathcal{U}}_2$. $\tilde{\mathcal{U}}_1 + \tilde{\mathcal{U}}_2$ has sequences with one $\tilde{\mathcal{U}}_1$, three $\tilde{\mathcal{U}}_2$, and two $\tilde{\mathcal{U}}_1$.

C. Experimental validation

Two measurements are used for comparison: *precision* measures how consistent results are across repetitions; *accuracy* considers closeness of the mean of a set of measurements to the actual goal.

Our tree was built so that the proportion of $\tilde{\mathcal{U}}_0$, $\tilde{\mathcal{U}}_1$ and $\tilde{\mathcal{U}}_2$ were equally distributed over 250,000 nodes. The start location is $X_S = (-100\ cm, 100\ cm, 0\ rad)$ and the goal location is $X_G = (100\ cm, -100\ cm, \pi\ rad)$. To simplify our experiments, we assume that the object rotates in a clockwise direction, so all motion primitives have a fixed direction of rotation.

Fig. 9 shows the three different planners *simple*, *adjustable*, and *adjustable + replanning* with four preferences as shown in $\tilde{\mathcal{U}}_0$-*only* in first column, $\tilde{\mathcal{U}}_1$-*only* in second column, $\tilde{\mathcal{U}}_1$+$\tilde{\mathcal{U}}_2$ in third column, $\tilde{\mathcal{U}}_2$-*only* in fourth column. Ten trials are shown for each case. The first column of Fig. 9 shows the initial state, and we can see a representative initialization motion (in green) for each primitive.

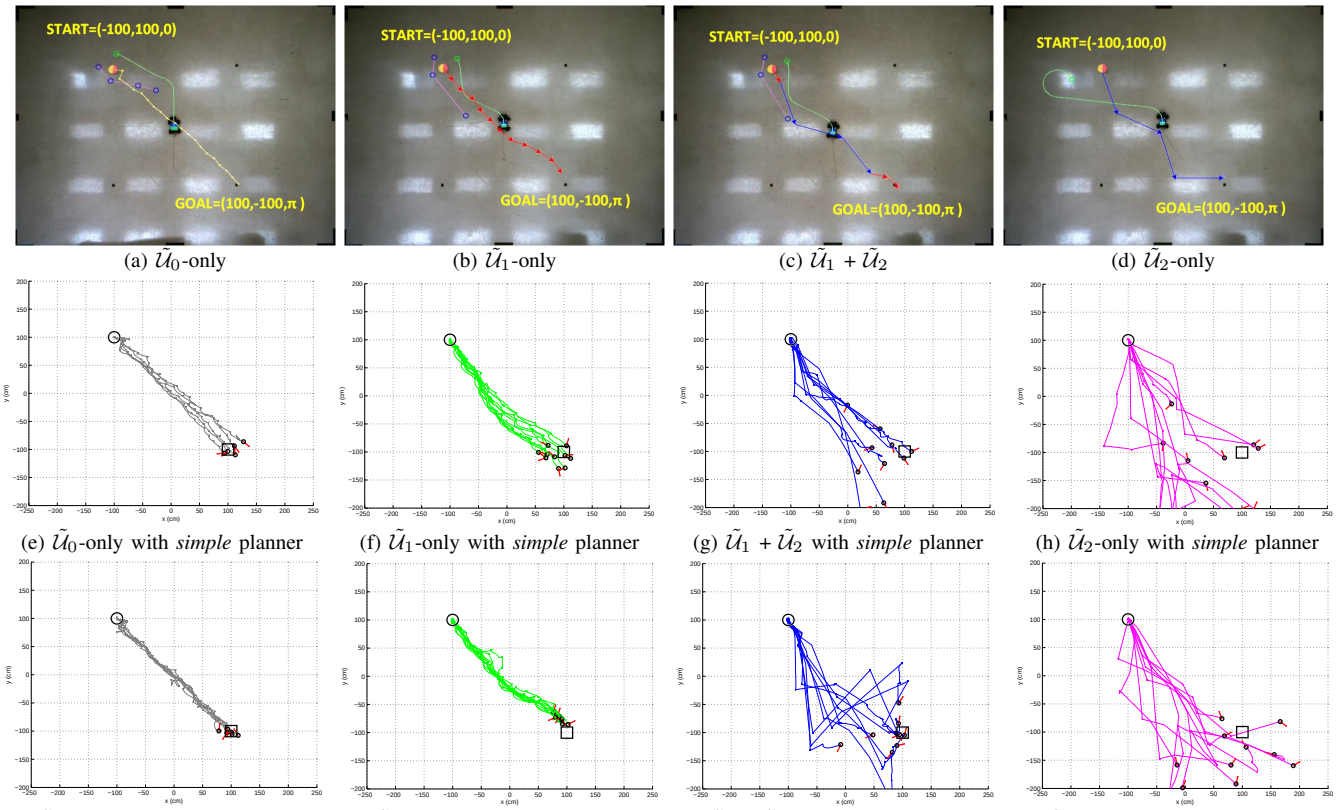
The second row of Fig. 9 shows the *simple* planner. The third row of Fig. 9 shows the *adjustable* planner. We can see that the *adjustable* planner has increased precision compared to the *simple* planner. From left to right, we see that accuracy also increases. From the third and fourth column of Fig. 9, we might think that $\tilde{\mathcal{U}}_2$ alone is useless. However, those preferences can be effective with the *adjustable + replanning* planner; applying the *adjustable + replanning* planner for $\tilde{\mathcal{U}}_1$+$\tilde{\mathcal{U}}_2$ gives the (representative) result magnified in Fig. 10. The replanning phase greatly increases accuracy. The replanning phase takes considerable computational time but certainly improves the quality of both $\tilde{\mathcal{U}}_2$-*only* and $\tilde{\mathcal{U}}_1$+$\tilde{\mathcal{U}}_2$.

A detailed analysis of all the results appears in Fig. 11. Fig. 11(a) shows the model's prediction (blue), the *simple* planner (green), the *adjustable* planner (red), and the *replanning* planner (magenta). The models consistently underestimate execution cost, likely due to difficulties in controlling the RC car robustly. The estimate's deficiencies were traced back to costs in the initialization steps for the primitives.When errors are big enough, compared to given trajectories, re-planning the robot path takes added time. Primitive $\tilde{\mathcal{U}}_0$ especially has a substantial gap between the model and reality, owing to the reversing motions further exacerbated by a sequence of more than twenty $\tilde{\mathcal{U}}_0$s.

From Fig. 11(b), adjustment and/or replanning reduce distance errors. The bar shows the mean square error, the error-bars depict the maximum and the minimum values.

The additional execution time for replanning in Fig. 11(a) comes from the execution time of additional motion primitives. Note that the times reported for replanning do not consider the time necessary for generating the tree, as they only reflect execution costs not computational ones.

Next, we briefly summarize results exploring cost preferences in the environment with obstacle collisions. In this case, we use α_2 for the scaled cost function. We have



(a) $\tilde{\mathcal{U}}_0$-only (b) $\tilde{\mathcal{U}}_1$-only (c) $\tilde{\mathcal{U}}_1 + \tilde{\mathcal{U}}_2$ (d) $\tilde{\mathcal{U}}_2$-only

(e) $\tilde{\mathcal{U}}_0$-only with *simple* planner (f) $\tilde{\mathcal{U}}_1$-only with *simple* planner (g) $\tilde{\mathcal{U}}_1 + \tilde{\mathcal{U}}_2$ with *simple* planner (h) $\tilde{\mathcal{U}}_2$-only with *simple* planner

(i) $\tilde{\mathcal{U}}_0$-only with *adjustable* planner (j) $\tilde{\mathcal{U}}_1$-only with *adjustable* planner (k) $\tilde{\mathcal{U}}_1 + \tilde{\mathcal{U}}_2$ with *adjustable* planner (l) $\tilde{\mathcal{U}}_2$-only with *adjustable* planner

Fig. 9. An overview of our results. The small circles near the goal location indicate the final object states (orientation as a red line).

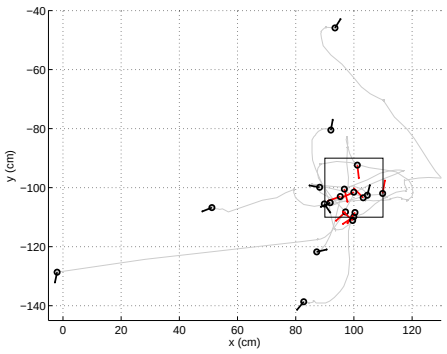
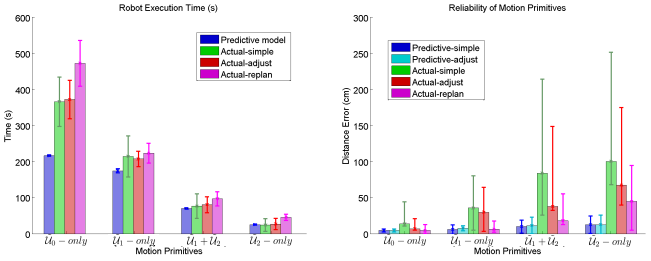


Fig. 10. We apply the replanning phase after the result of Fig. 9(k). Ten scattered objects are finally moved to the near goal location with good accuracy and precision.



(a) Primitive execution times (mean and standard deviation). (b) Error as distance from goal (mean, min, and max).

Fig. 11. The data in Fig. 9 summarized.

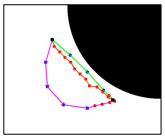
three preferences, each shown in Fig. 12. Path-1 ($(\alpha_1, \alpha_3) = (0.0003, 0.1)$) consists of three $\tilde{\mathcal{U}}_2$s and two $\tilde{\mathcal{U}}_1$s. Path-2 ($(\alpha_1, \alpha_3) = (0.0003, 0.4)$) has four $\tilde{\mathcal{U}}_2$s and three $\tilde{\mathcal{U}}_1$s. Path-3 ($(\alpha_1, \alpha_3) = (0.0001, 0.1)$) has twelve $\tilde{\mathcal{U}}_1$s. Values are reported for 10 trials for each path. Path-1 is the fastest method, but it highest risk of collision (at 40 %). Path-2 has a reasonable execution time and no collision. Path-3 is the most conservative path.

D. Discussion

1) Implementation challenges: An issue we encountered with our implementation was that it was difficult to assure that the robot maintained a fixed velocity. This is because friction varies depending on whether the object (and tail, including just some segments of it) are moving or not. A consequence is that the total execution time reported above is computed based on the average velocity of the robot.

2) Investigation of practical scenarios: Dynamics-based motions, such as $\tilde{\mathcal{U}}_2$, are not merely a scientific curiosity, they can be useful in several practical scenarios. When a robot cannot access a narrow passage, a high-speed striking motion that can move an object quickly and reliably through the passage. If an object is stuck near a wall or a corner, the robot can use $\tilde{\mathcal{U}}_2$ to manipulate a stuck object directing it outwards from afar. It is also potentially possible for a robot with a tail to use $\tilde{\mathcal{U}}_2$ to separate multiple gathered objects, so that each object can be brought into their desired location, much like a billiards break.

Our approach is not limited to cylindrical objects. Owing to space limitations, we do not report all our data here, but experiments showed that objects of various shapes (including a square, a triangle, a star) with convex hulls of comparable perimeters (and approximate location of center of mass and center of friction) work with the same parameterized models we have described for the cylinder. The approach is fairly oblivious to object geometry.



Motion Primitive	Arrival time (s)	Collision rate	Arrival rate
Path 1 (green)	(84.2, 9.17)	40%	60%
Path 2 (purple)	(110, 10)	0%	80%
Path 3 (red)	(360, 15)	0%	100%

Fig. 12. Analysis of the preferences in the obstacle environment. The black area is the static obstacle area. Reported arrival time (seconds) is (μ, σ) for 10 trials of each path. We count the number of collision. We also count the number of arrival at the destination (± 30 cm radius).

VII. Conclusion

Even though rope-like structures are simple, cheap, and versatile tools, compliant passive tails have received little attention for robotic manipulation. We speculate that this is a consequence of the modelling difficulties inherent to such systems; this paper's response to the challenge is to use a collection of models of varying verisimilitude. We have demonstrated how the dynamics of a compliant, unactuated tail can be successfully exploited for manipulation. We have shown the effectiveness of an approach that uses simple stochastic models, each associated with a structured primitive. The primary planning technology used was sampling-based motion planning with a particle-based representation for error propagation. We demonstrated our system with physical robot experiments and the results show that various preferences can be expressed effectively, ultimately being reflected in different mixes of primitives being executed.

Appendix A
Learning Model Parameters

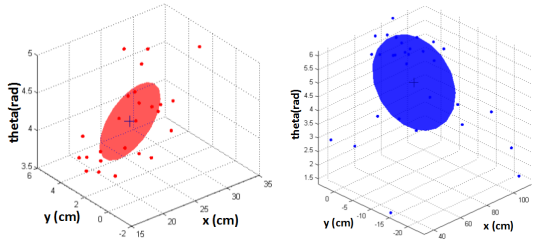
Our motion primitives have several parameters. Primitive $\tilde{\mathcal{U}}_0$ has an analytical model (but does include a small variance). For $\tilde{\mathcal{U}}_0$, we only need to know L_{min}, which is approximately 20 cm in our tests. However, $\tilde{\mathcal{U}}_1$ and $\tilde{\mathcal{U}}_2$ are much more complicated, so we resorted to collecting data. We placed the robot and tail in random initial configurations, then we executed the primitives over 20 times while recording data.

The following summarize the collected parameters through the repeated execution of each motion primitive in Fig. 13.

$$X_{mean_1} = \begin{bmatrix} 26.9 \\ 5.23 \\ 4.899 \end{bmatrix} \quad \Sigma_1 = \begin{bmatrix} 5.1673 & 1.6611 & 0.2215 \\ 1.6611 & 1.4337 & 0.1414 \\ 0.2215 & 0.1414 & 0.0183 \end{bmatrix}$$

$$X_{mean_3} = \begin{bmatrix} 80.04 \\ -4.57 \\ 4.51 \end{bmatrix} \quad \Sigma_3 = \begin{bmatrix} 122 & 5.6 & 0 \\ 5.6 & 6.99 & 0.902 \\ 0 & 0.902 & 0.4183 \end{bmatrix}$$

Σ_2 and Σ_4 are taken as $[3, 0.1, 0]$, which are measured directly via execution of a Dubins curve.



(a) $\tilde{\mathcal{U}}_1$ (b) $\tilde{\mathcal{U}}_2$

Fig. 13. The object is located at (0,0,0), and the robot executes each motion primitive with $\phi = 0$. Settings match those in Figs 5 and 6.

References

[1] K. Lynch and M. Mason, "Stable Pushing: Mechanics, Controllability, and Planning," *Int. J. of Robotics Research*, vol. 15, pp. 533–556.

[2] T. Meriçli, M. Veloso, and H. L. Akin, "Push-manipulation of complex passive mobile objects using experimentally acquired motion models," *Autonomous Robots*, vol. 38, pp. 317–329, 2015.

[3] J. Fink, M. A. Hsieh, and V. Kumar, "Multi-Robot Manipulation via Caging in Envirionments with Obstacles," in *Proc. of Int. Conf. on Robotics and Automation*, 2008.

[4] W. H. Huang, E. P. Krotkov, and M. T. Mason, "Impulsive Manipulation," in *Proc. of Int. Conf. on Robotics and Automation*, May 1995.

[5] B. Donald, L. Gariepy, and D. Rus, "Distibuted Manipulation of Multiple Objects using Ropes," in *Proc. of Int. Conf. on Robotics and Automation*, 2000.

[6] S. Bhattacharya, S. Kim, H. Heidarsson, G. Sukhatme, and V. Kumar, "A topological approach to using cables to separate and manipulate sets of objects," *Int. J. of Robotics Research*, pp. 1–17, 2015.

[7] G. Robinson and J. Davies, "Continuum Robots - A State of the Art," in *Proc. of Int. Conf. on Robotics and Automation*, 1999.

[8] S. Chiaverini, G. Oriolo, and I. D. Walker, "Kinematically Redundant Manipulators," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer-Verlag Heidelberg, 2008, ch. 11.

[9] R. J. W. III and B. A. Jones, "Design and Kinematic Modeling of Constant Curvature Continuum," *Int. J. of Robotics Research*, vol. 29, pp. 1661–1683, 2010.

[10] L. S. Cowan and I. D. Walker, "The Importance of Continuous and Discrete Elements in Continuum Robots," *International Journal of Advanced Robotic Systems*, vol. 10, pp. 1–13, 2013.

[11] M. R. Dogar and S. S. Srinivasa, "A Planning Framework for Non-Prehensile Manipulation under Clutter and Uncertainty," *Autonomous Robots*, vol. 33, no. 3, pp. 217–236, 2012.

[12] M. Phillips, B. Cohen, S. Chitta, and M. Likhachev, "E-graphs: Bootstrapping planning with experience graphs," in *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[13] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *Proceedings of International Conference on Robotics and Automation*, Saint Paul, Minnesota, USA, May 2012.

[14] A. Bry and N. Roy, "Rapidly-exploring Random Belief Trees for Motion Planning Under Uncertainty," in *Proc. of Int. Conf. on Robotics and Automation*, 2011.

[15] T. Libby, T. Y. Moore, E. Chang-Siu, D. Li, D. J. Cohen, A. Jusufi, and R. J. Full, "Tail-assisted pitch control in lizards, robots and dinosaurs," *Nature Letter*, vol. 481, pp. 181–186, 2012.

[16] E. Chang-Siu, T. Libby, M. Tomizuka, and R. J. Full, "A lizard-Inspired Active Tail Enables Rapid Maneuvers and Dynamic Stabilization in a Terretrial Robot," in *Proc. of Int. Conf. on Intelligent Robots and Syst.*, 2011.

[17] J. Zhao, T. Zhao, N. Xi, F. J. Cintron, M. W. Mutka, and L. Xiao, "Controlling Aerial Maneuvering of a Miniature Jumping Robot Using Its Tail," in *Proc. of Int. Conf. on Intelligent Robots and Syst.*, 2013.

[18] W. S. Rone and P. Ben-Tzvi, "Continuum Robotic Tail Loading Analysis for Mobile Robot Stabilization and Maneuvering," in *Proceedings of IDETC/CIE*, Buffalo, New York, USA, Aug. 2014.

[19] A. Patel and M. Brrae, "Rapid Turning at High-Speed: Inpirations from the Cheetah's Tail," in *Proc. of the International Conference on Intelligent Robots and Systems*, 2013.

[20] N. J. Kohut, A. O. Pullin, D. W. Haldane, D. Zarrouk, and R. S. Fearing, "Precise Dynamic Turning of a 10 cm Legged Robot on a Low Friction Surface Using a Tail," in *Proc. of Int. Conf. on Robotics and Automation*, 2013.

[21] R. Briggs, J. Lee, M. Haberland, and S. Kim, "Tails in Biomimetic Design: Analysis, Simulation, and Experiment," in *Proceedings of the International Conference on Intelligent Robots and Systems*, Vilamoura, Algarve, Portugal, Oct. 2012.

[22] J. Ackerman, X. Da, and J. Seipel, "Mobility of Legged Robot Locomotion with Elastically-suspended Loads over Rough Terrain," in *Proc. of CLAWAR*, 2012.

[23] G.-H. Liu, H.-Y. Lin, H.-Y. Lin, S.-T. Chen, and P.-C. Lin, "A Bio-Inspired Hopping Kangaroo Robot with An Active Tail," *Journal of Bionic Engineering*, pp. 541–555, Oct. 2014.

[24] P. R. Giordano and M. Vendittelli, "Shortest Paths to Obstacles for a Polygonal Dubins Car," *IEEE Transactions on Robotics*, vol. 25, pp. 1184–1191, 2009.

[25] S. M. LaValle and J. James J. Kuffner, "Randomized Kinodynamic Planning," *Int. J. of Robotics Research*, vol. 20, pp. 378–400, 2001.