Bandit-Based Model Selection for Deformable Object Manipulation

Dale M^{c}Conachie and Dmitry Berenson

University of Michigan, Ann Arbor MI 48109, USA
dmcconac@umich.edu, berenson@eecs.umich.edu

Abstract. We present a novel approach to deformable object manipulation that does not rely on highly-accurate modeling. The key contribution of this paper is to formulate the task as a Multi-Armed Bandit problem, with each arm representing a model of the deformable object. To "pull" an arm and evaluate its utility, we use the arm's model to generate a velocity command for the gripper(s) holding the object and execute it. As the task proceeds and the object deforms, the utility of each model can change. Our framework estimates these changes and balances exploration of the model set with exploitation of high-utility models. We also propose an approach based on Kalman Filtering for Non-stationary Multi-armed Normal Bandits (KF-MANB) to leverage the coupling between models to learn more from each arm pull. We demonstrate that our method outperforms previous methods on synthetic trials, and performs competitively on several manipulation tasks in simulation.

1 Introduction

One of the primary challenges in manipulating deformable objects is the difficulty of modeling and simulating them. The most common simulation methods use Mass-Spring models [1, 2], which are generally not accurate for large deformations [3], and Finite-Element models [4, 5], which require significant tuning and are very sensitive to the discretization of the object. Approaches like [6, 7] bypass this challenge by using offline demonstrations to teach the robot specific manipulation tasks; however, when a new task is attempted a new training set needs to be generated. In our application we are interested in a way to manipulate a deformable object without a high-fidelity model or training set available *a priori*. For instance, imagine a robot encountering a new piece of clothing for a new task. While it may have models for previously-seen clothes or training sets for previous tasks, there is no guarantee that those models or training sets are appropriate for the new task. Also, depending on the state of the clothing different models may be most useful at different times in the manipulation task.

Rather than assuming we have a high-fidelity model of a deformable object interacting with its environment, our approach is to have multiple models available for use, any one of which may be useful at a given time. We do not assume these models are correct, we simply treat the models as having some measurable *utility* to the task. The *utility* of a given model is the expected reduction in task error when using this model to generate robot motion. As the task proceeds, the utility of a given model may change, making other models more suitable for the current part of the task. However, without testing a model's prediction, we do not know its true utility. Testing every model in the set is impractical, as all models would need to be tested at every step, and performing a test changes the state of the object and may drive it into a local minimum. The key question is then which model should be selected for testing at a given time.

The central contribution of this paper is framing the model selection problem as a Multi-Armed Bandit (MAB) problem where the goal is to find the model that has the highest utility for a given task. An arm represents a single model of the deformable object; to "pull" an arm is to use the arm's model to generate and execute a velocity command for the robot. The reward received is the reduction in task error after executing the command. In order to determine which model has the highest utility we need to explore the model space, however we also want to exploit the information we have gained by using models that we estimate to have high utility. One of the primary challenges in performing this exploration versus exploitation trade-off is that our models are inherently coupled and non-stationary; performing an action changes the state of the system which can change the utility of every model, as well as the reward of pulling each arm. While there is work that frames robust trajectory selection as a MAB problem [8], we are not aware of any previous work which either 1) frames model selection for deformable objects as a MAB problem; or 2) addresses the coupling between arms for non-stationary MAB problems.

In our experiments, we show how to formulate a MAB problem with coupled arms for Jacobian-based models. We perform our experiments on three synthetic systems, and on three deformable object manipulation tasks in the Bullet [9] simulator. We demonstrate that formulating model selection as a MAB problem is able to successfully perform all three manipulation tasks. We also show that our proposed MAB algorithm outperforms previous MAB methods on synthetic trials, and performs competitively on the manipulation tasks.

2 Related Work

Deformable Object Modeling: One of the key challenges in manipulating deformable objects is the difficulty inherent in modeling and simulating them. While there has been some progress towards online modeling of deformable objects [10, 11] these methods rely on a time consuming training phase for each object to be modeled. Of particular interest are Jacobian-based models such as [12] and [13]. In these models we assume that there is some function $F : SE(3)^G \to \mathbb{R}^N$ which maps a configuration of G robot grippers $q \in SE(3)^G$ to a parameterization of the deformable object $\mathcal{P} \in \mathbb{R}^N$, where N is the dimensionality of the parameterization of the deformable object. These models are then linearized by calculating an approximation of the the Jacobian of F:

$$\begin{aligned}\mathcal{P} &= F(q)\\ \frac{\partial \mathcal{P}}{\partial t} &= \frac{\partial F(q)}{\partial q}\frac{\partial q}{\partial t}\\ \dot{\mathcal{P}} &= J(q)\dot{q}\ . \end{aligned} \tag{1}$$

Computation of an exact Jacobian $J(q)$ at a given configuration q is often computationally intractable and requires high-fidelity models and simulators, so instead approximations are frequently used. A shared characteristic of these approximations is some reliance on tuned parameters. This tuning process can be tedious, and in some cases needs to be done on a per-task basis.

In this paper we consider two types of approximate Jacobian models. The first approximation we use is a *diminishing-rigidity Jacobian* [12] which assumes that points on the deformable object that are near a gripper move "almost rigidly" with respect to the gripper while points that are further away move "less rigidly". This approximation uses deformability parameters to control how quickly the rigidity decreases with distance. The second approximation we use is an *adaptive Jacobian* [13] which uses online estimation to approximate $J(q)$. Adaptive Jacobian models rely on a learning rate to control how quickly the estimation changes from one timestep to the next.

Model Selection: In order to accomplish a given manipulation task, we need to determine which type of model to use at the current time to compute the next velocity command, as well as how to set the model parameters. Frequently this selection is done manually, however, there are methods designed to make these determinations automatically. Machine learning techniques such as [14, 15] rely on supervised training data in order to intelligently search for the best regression or classification model, however, it is unclear how to acquire such training data for the task at hand without having already performed the task. The most directly applicable methods come from the Multi-Armed Bandit (MAB) literature [16–18]. In this framework there are multiple actions we can take, each of which provides us with some reward according to an unknown probability distribution. The problem then is to determine which action to take (which arm to pull) at each time step in order to maximize reward.

The MAB approach is well-studied for problems where the reward distributions are *stationary*; i.e. the distributions do not change over time [16, 19]. This is not the case for deformable object manipulation; consider the situation where the object is far away from the goal versus the object being at the goal. In the first case there is a possibility of an action moving the object closer to the goal and thus achieving a positive reward; however, in the second case any motion would, at best, give zero reward.

Recent work [20] on non-stationary MAB problems offer promising results that utilize independent Kalman filters as the basis for the estimation of a non-stationary reward distribution for each arm. This algorithm (KF-MANB) provides a Bayesian estimate of the reward distribution at each timestep, assuming that the reward is normally distributed. KF-MANB then performs Thompson sampling [19] to select which arm to pull, choosing each in proportion to the belief that it is the optimal arm. We build on this approach in this paper to produce a method that also accounts for dependencies between arms by approximating the coupling between arms at each timestep.

For the tasks we address, the reward distributions are both non-stationary as well as *dependent*. Because all arms are operating on the same physical system, pulling one arm both gives us information about the distributions over other arms, as well as changing the future reward distributions of all arms. While work has been done on dependent bandits [21, 22], we are not aware of any work addressing the combination of non-stationary and dependent bandits. Our method for model selection is inspired by KF-MANB, however we directly use coupling between models in order to form a joint reward distribution over all models. This enables a pull of a single arm to provide information about all arms, and thus we spend less time exploring the model space and more time exploiting useful models to perform the manipulation task.

3 Problem Statement

Let the robot be represented by a set of G grippers with configuration $q \in SE(3)^G$. We assume that the robot configuration can be measured exactly; in this work we assume the robot to be a set of free floating grippers; in practice we can track the motion of these with inverse kinematics on a real robot. We use the Lie algebra [23] of $SE(3)$ to represent robot gripper velocities. This is the tangent space of $SE(3)$, denoted as $\mathfrak{se}(3)$. The velocity of a single gripper g is then $\dot{q}_g = \begin{bmatrix} v_g^T & \omega_g^T \end{bmatrix}^T \in \mathfrak{se}(3)$ where v_g and ω_g are the translational and rotational components of the gripper velocity. We define the velocity of the entire robot to be $\dot{q} = \begin{bmatrix} \dot{q}_1^T & \dots & \dot{q}_G^T \end{bmatrix}^T \in \mathfrak{se}(3)^G$. We define the inner product of two gripper velocities $\dot{q}_1, \dot{q}_2 \in \mathfrak{se}(3)$ to be $\langle \dot{q}_1, \dot{q}_2 \rangle = \langle \dot{q}_1, \dot{q}_1 \rangle_c = v_1^T v_2 + c\omega_1^T \omega_2$, where c is a non-negative scaling factor relating rotational and translational velocities.

The configuration of a deformable object is a set $\mathcal{P} \subset \mathbb{R}^3$ of P points. We assume that we have a method of sensing $\mathcal{P}$. To measure the norm of a deformable object velocity $\dot{\mathcal{P}} = \begin{bmatrix} \dot{\mathcal{P}}_1^T & \dots & \dot{\mathcal{P}}_P^T \end{bmatrix}^T \in \mathbb{R}^{3P}$ we will use a weighted Euclidean norm

$$\|\dot{\mathcal{P}}\|_W^2 = \sum_{i=1}^{P} w_i \dot{\mathcal{P}}_i^T \dot{\mathcal{P}}_i = \dot{\mathcal{P}}^T \operatorname{diag}(W) \dot{\mathcal{P}} \tag{2}$$

where $W = \begin{bmatrix} w_1 & \dots & w_P \end{bmatrix}^T \in \mathbb{R}^P$ is a set of non-negative weights. The rest of the environment is denoted $\mathcal{O}$ and is assumed to be both static, and known exactly.

Let a *deformation model* be defined as a function $\phi : \mathfrak{se}(3)^G \to \mathbb{R}^{3P}$ which maps a change in robot configuration $\dot{q}$ to a change in object configuration $\dot{\mathcal{P}}$. Let $\mathcal{M}$ be a set of M deformable models which satisfy this definition. Each model is associated with a robot command function $\psi : \mathbb{R}^{3P} \times \mathbb{R}^P \to \mathfrak{se}(3)^G$ which maps a desired deformable object velocity $\dot{\mathcal{P}}$ and weight W (Sec. 5.2) to a robot velocity command $\dot{q}$. ϕ and ψ also take the object and robot configuration

$(\mathcal{P}, q)$ as additional input, however this is omitted for clarity. When a model m is selected for testing, the model generates a gripper command

$$\dot{q}_m(t) = \psi_m(\dot{\mathcal{P}}(t), W(t)) \tag{3}$$

which is then executed for one unit of time, moving the deformable object to configuration $\mathcal{P}(t+1)$.

The problem we address in this paper is which model $m \in \mathcal{M}$ to select in order to to move G grippers such that the points in $\mathcal{P}$ align as closely as possible with some task-defined set of T target points $\mathcal{T} \subset \mathbb{R}^3$, while avoiding gripper collision and excessive stretching of the deformable object. Each task defines a function ρ which measures the alignment error between $\mathcal{P}$ and $\mathcal{T}$. The method we present is a local method which picks a single model m_* at each timestep to treat as the true model. This model is then used to reduce error as much as possible while avoiding collision and excessive stretching.

$$m_* = \operatorname*{argmin}_{m \in \mathcal{M}} \rho(\mathcal{T}, \mathcal{P}(t+1)) \tag{4}$$

We show that this problem can be treated as an instance of the multi-arm non-stationary dependent bandit problem.

4 Bandit-Based Model Selection

The primary difficulty with solving (4) directly is that the effectiveness of a particular model in minimizing error is unknown. It may be the case that no model in the set produces the optimal option, however, this does not prevent a model from being useful. In particular the *utility* of a model may change from one task to another, and from one configuration to another as the deformable object changes shape, and moves in and out of contact with the environment. We start by defining the utility $u_m(t) \in \mathbb{R}$ of a model as the expected improvement in task error ρ if model m is used to generate a robot command at time t. If we know which model has the highest utility then we can solve (4). This leads to a classic exploration versus exploitation trade-off where we need to explore the space of models in order to learn which one is the most useful, while also exploiting the knowledge we have already gained. The multi-armed bandit framework is explicitly designed to handle this trade-off.

In the MAB framework, each arm represents a model in $\mathcal{M}$; to pull arm m is to command the grippers with velocity $\dot{q}_m(t)$ (Eq. 3) for 1 unit of time. We then define the *reward* $r_m(t+1)$ after taking action $\dot{q}_m(t)$ as the improvement in error

$$r_m(t+1) = \rho(t) - \rho(t+1) = u_m(t) + w \tag{5}$$

where w is a zero-mean noise term. The goal is to pick a sequence of arm pulls to minimize total expected regret $R(T_f)$ over some (possibly infinite) horizon T_f

$$E[R(T_f)] = \sum_{t=1}^{T_f} (E[r^*(t)] - E[r(t)]) \tag{6}$$

where $r^*(t)$ is the reward of the best model at time t. The next section describes how to use bandit-based model selection for deformable object manipulation.

5 MAB Formulation for Deformable Object Manipulation

Algorithm 1 MainLoop($\mathcal{O}, \beta, \lambda$)

```
 1: $t \leftarrow 0$
 2: $D \leftarrow$ GeodesicDistanceMatrix($\mathcal{P}_{relaxed}$)
 3: $\mathcal{M} \leftarrow$ InitializeModels($D$)
 4: InitialzeBanditAlgorithm()
 5: $\mathcal{P}(0) \leftarrow$ SensePoints()
 6: $q(0) \leftarrow$ SenseRobotConfig()
 7: while true do
 8:     $m \leftarrow$ SelectArmUsingBanditAlgorithm()
 9:     $\mathcal{T} \leftarrow$ GetTargets()
10:     $\dot{\mathcal{P}}_e, W_e \leftarrow$ ErrorCorrection($\mathcal{P}(t), \mathcal{T}$)
11:     $\dot{\mathcal{P}}_s, W_s \leftarrow$ StretchingCorrection($D, \lambda, \mathcal{P}(t)$)
12:     $\dot{\mathcal{P}}_d, W_d \leftarrow$ CombineTerms($\dot{\mathcal{P}}_e, W_e, \dot{\mathcal{P}}_s, W_s$)
13:     $\dot{q}_d \leftarrow \psi_m(\dot{\mathcal{P}}_d, W_d)$
14:     $\dot{q} \leftarrow$ ObstacleRepulsion($\dot{q}_d, \mathcal{O}, \beta$)
15:     CommandConfiguration($q(t) + \dot{q}$)
16:     $\mathcal{P}(t+1) \leftarrow$ SensePoints()
17:     $q(t+1) \leftarrow$ SenseRobotConfig()
18:     UpdateBanditAlgorithm()
19:     $t \leftarrow t + 1$
20: end while
```

Our algorithm (Alg. 1) can be broken down into four major sections and an initialization block. In the initialization block we pre-compute the geodesic distance between every pair of points in $\mathcal{P}$ when the deformable object is in its "natural" or "relaxed" state and store the result in D. These distances are used to construct the deformation models (Sec. 5.3), as well as to avoid overstretching the object (Sec. 5.2). At each iteration we: 1) pick a model to use to achieve the desired direction (Sec. 5.1); 2) compute the task-defined desired direction to move the deformable object (Sec. 5.2); 3) generate a velocity command using the chosen model (Sec. 5.3); 4) modify the command to avoid obstacles (Sec. 5.2); and 5) update bandit algorithm parameters (Sec. 5.1).

5.1 Algorithms for MAB

Previous solutions [16, 20] to minimizing (6) assume that rewards for each arm are normally and independently distributed and then estimate the mean and variance of each Gaussian distribution. We test three algorithms in our experiments: Upper Confidence Bound for normally distributed bandits UCB1-Normal, Kalman Filter Based Solution to Non-Stationary Multi-arm Normal Bandits (KF-MANB), and our extension of KF-MANB, Kalman Filter Based Solution to Non-Stationary Multi-arm Normal Dependent Bandit (KF-MANDB).

UCB1-Normal: The UCB1-Normal algorithm [16] treats each arm (model) as independent, estimating an optimistic Upper Confidence Bound (UCB) for the utility of each model. The model with the highest UCB is used to command the robot at each timestep. This algorithm assumes that the utility of each model is stationary, gradually shifting from exploration to exploitation as more information is gained. While our problem is non-stationary and dependant, we use

UCB1-Normal as a baseline algorithm to compare against due to its prevalence in previous work. The algorithm is shown in App. A.1 for reference.

KF-MANB: The Kalman Filter Based Solution to Non-Stationary Multi-arm Bandit (KF-MANB) algorithm [20] uses independent Kalman filters to estimate the utility distribution of each model, and then uses Thompson sampling [19] to chose which model to use at each timestep. Because this algorithm explicitly allows for non-stationary reward distributions, it is able to "switch" between models much faster than UCB1-Normal. The KF-MANB algorithm is shown in App. A.1 for reference.

KF-MANDB: We also propose a variant of KF-MANB, replacing the independent Kalman filters with a single joint Kalman filter. This enables us to capture the correlations between models, allowing us to learn more from each pull. We start by defining utility as a linear system with Gaussian noise with process model $u(t+1) = u(t) + v$ and observation model $r(t) = Cu(t) + w$ where $u(t)$ is our current estimate of the relative utility of each model, while v and w are zero-mean Gaussian noise terms. C is a row vector with a 1 in the column of the model we used and zeros elsewhere. The variance on w is defined as $\sigma_{obs}^2 \eta^2$. η is a tuning parameter to scale the covariance to match the reward scale of the specific task, while σ_{obs} controls how much we believe each new observation.

To define the process noise v we want to leverage correlations between models; if two model predictions are similar, the utility of these models is likely correlated. To measure the similarity between two models i and j we use the angle between their gripper velocity commands $\dot{q}_i$ and $\dot{q}_j$. This similarity is then used to directly construct a covariance matrix for each arm pull:

$$\begin{aligned} v &\sim \mathcal{N}\left(0, \sigma_{tr}^2 \eta^2 (\xi \Sigma + (1-\xi)\,\mathbf{I})\right) \\ \Sigma_{i,j} &= \frac{\langle \dot{q}_i, \dot{q}_j \rangle}{\|\dot{q}_i\| \|\dot{q}_j\|} = \cos\theta_{i,j} \ . \end{aligned} \tag{7}$$

σ_{tr} is the standard Kalman Filter transition noise factor tuning parameter. $\xi \in [0,1]$ is the correlation strength factor; larger ξ gives more weight to the arm correlation, while smaller ξ gives lower weight. When ξ is zero then KF-MANDB will have the same update rule as KF-MANB, thus we can view KF-MANDB as a generalizion of KF-MANB, allowing for correlation between arms.

After estimating the utility of each model and the noise parameters at the current timestep, these values are then passed into a Kalman filter which estimates a new joint distribution. The next step is the same as KF-MANB; we draw a sample from the resulting distribution, then use the model that yields the largest sample to generate the next robot command. In this way we automatically switch between exploration and exploitation as the system evolves; if we are uncertain of the utility of our models then we are more likely to choose different models from one timstep to the next. If we believe that we have accurate estimates of utility, then we are more likely to choose the model with the highest utility.

5.2 Determining $\dot{q}$

Error Correction We build on previous work [12], splitting the desired deformable object movement into two parts: an error correction part and a stretching correction part. When defining the direction we want to move the deformable object to minimize error we calculate two values; which direction to move the deformable object points $\dot{\mathcal{P}}_e$ and the importance of moving each deformable object point W_e. This is analogous to computing the gradient of error, as well as an “importance factor” for each part of the gradient. We need these weights to be able to differentiate between points of the object where the error function is a plateau versus points where the error function is at a local minimum (Fig. 1). Typically this is achieved using a Hessian, however our error function does not have a second derivative at many points. We use the `ErrorCorrection` (Alg. 2) function to calculate these values. Each target point $\mathcal{T}_i \in \mathcal{T}$ defines a potential field, pulling the nearest point on the deformable object $\mathcal{P}_k$ towards $\mathcal{T}_i$. W_e is set to the maximum distance $\mathcal{P}_k$ is being pulled by any target point. This allows W_e to be insensitive to changes in discretization.

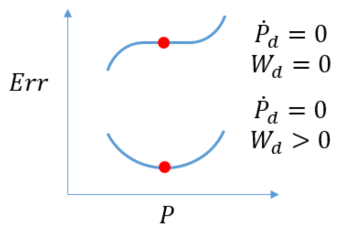

Fig. 1. Top Line: moving the point does not change the error, thus the desired movement is zero, however, it is not important to achieve zero movement, thus $W_d = 0$. Bottom Line: error is at a local minimum; thus moving the point increases error.

Algorithm 2 ErrorCorrection($\mathcal{P}, \mathcal{T}$)

```
1: $\dot{\mathcal{P}}_e \leftarrow \mathbf{0}_{3P\times 1}$, $W_e \leftarrow \mathbf{0}_{P\times 1}$
2: for $i \in \{1, 2, \dots, T\}$ do
3:     $k \leftarrow \mathrm{argmin}_{j\in\{1,2,\dots,P\}} \|\mathcal{T}_i - \mathcal{P}_j\|$
4:     $\dot{\mathcal{P}}_{e,k} \leftarrow \dot{\mathcal{P}}_{e,k} + \mathcal{T}_i - \mathcal{P}_k$
5:     $W_{e,k} \leftarrow \max(W_{e,k}, \|\mathcal{T}_i - \mathcal{P}_k\|)$
6: end for
7: return $\{\dot{\mathcal{P}}_e, W_e\}$
```

Algorithm 3 StretchingCorrection($D, \lambda, \mathcal{P}$)

```
1: $E \leftarrow$ EuclidianDistanceMatrix($\mathcal{P}$)
2: $\dot{\mathcal{P}}_s \leftarrow \mathbf{0}_{3P\times 1}$, $W_s \leftarrow \mathbf{0}_{P\times 1}$
3: $\Delta \leftarrow E - D$
4: for $i \in \{1, 2, \dots, P\}$ do
5:     for $j \in \{i+1, \dots, P\}$ do
6:         if $\Delta_{i,j} > \lambda$ then
7:             $v \leftarrow \Delta_{i,j}(\mathcal{P}_j - \mathcal{P}_i)$
8:             $\dot{\mathcal{P}}_{s,i} \leftarrow \dot{\mathcal{P}}_{s,i} + \frac{1}{2}v$
9:             $\dot{\mathcal{P}}_{s,j} \leftarrow \dot{\mathcal{P}}_{s,j} - \frac{1}{2}v$
10:            $W_{s,i} \leftarrow \max(W_{s,i}, \Delta_{i,j})$
11:            $W_{s,j} \leftarrow \max(W_{s,j}, \Delta_{i,j})$
12:        end if
13:    end for
14: end for
15: return $\{\dot{\mathcal{P}}_s, W_s\}$
```

Stretching Correction Our algorithm for stretching correction is similar to that found in [12], with the addition of a weighting term W_s, and a change in how we combine the two terms. We use the `StretchingCorrection` function (Alg. 3) to compute $\dot{\mathcal{P}}_s$ and W_s based on a task-defined stretching threshold $\lambda \geq 0$. First we compute the distance between every two points on the object and store the result in E. We then compare E to D which contains the relaxed lengths between every pair of points. If any two points are stretched by more than λ, we attempt to move the points closer to each other. We use the same strategy for setting the importance of this stretching correction W_s as we use for error correction. When combining stretching correction and error correction terms (Alg. 4) we prioritize stretching correction, accepting only the portion of the error correction that is orthogonal to the stretching correction term for each point.

Obstacle Avoidance In order to guarantee that the grippers do not collide with any obstacles, we use the same strategy from [12], smoothly switching between collision avoidance and other objectives (see Alg. 5). For every gripper g and an obstacle set $\mathcal{O}$ we find the distance d_g to the nearest obstacle, a unit vector $\dot{x}_{p_g}$ pointing from the obstacle to the nearest point on the gripper, and a Jacobian J_{p^g} between the gripper's DOF and the point on the gripper. The `Proximity` function is shown in Appendix C. $\beta > 0$ sets the rate at which we change between servoing and collision avoidance objectives. $\dot{q}_{\max,o} > 0$ is an internal parameter that sets how quickly we move the robot away from obstacles.

Algorithm 4 CombineTerms($\dot{\mathcal{P}}_e, W_e, \dot{\mathcal{P}}_s, W_s$)

```
1: for $i \in \{1, 2, \dots, P\}$ do
2:     $\dot{\mathcal{P}}_{d,i} \leftarrow \dot{\mathcal{P}}_{s,i} + \left(\dot{\mathcal{P}}_{e,i} - \mathrm{Proj}_{\dot{\mathcal{P}}_{s,i}} \dot{\mathcal{P}}_{e,i}\right)$
3:     $W_{d,i} \leftarrow W_{s,i} + W_{e,i}$
4: end for
5: return $\{\dot{\mathcal{P}}_d, W_d\}$
```

Algorithm 5 ObstacleRepulsion($\mathcal{O}, \beta$)

```
1: for $g \in \{1, 2, \dots, G\}$ do
2:     $J_{p^g}, \dot{x}_{p^g}, d_g \leftarrow \mathrm{Proximity}(\mathcal{O}, g)$
3:     $\gamma \leftarrow e^{-\beta d_g}$
4:     $\dot{q}_{c,g} \leftarrow J^{+}_{p^g} \dot{x}_{p^g}$
5:     $\dot{q}_{c,g} \leftarrow \frac{\dot{q}_{\max,o}}{\|\dot{q}_{c,g}\|} \dot{q}_{c,g}$
6:     $\dot{q}_g \leftarrow \gamma \left(\dot{q}_{c,g} + \left(\mathbf{I} - J^{+}_{p^g} J_{p^g}\right) \dot{q}_g\right) + (1-\gamma)\dot{q}_g$
7: end for
8: return $\dot{q}$
```

5.3 Jacobian Models

Every model must define a prediction function $\phi(\dot{q})$ and has an associated robot command function $\psi(\dot{\mathcal{P}}, W)$. This paper focuses on Jacobian-based models whose basic formulation Eq. (1) directly defines the deformation model ϕ

$$\phi(\dot{q}) = J\dot{q}. \tag{8}$$

When defining the robot command function ψ, we use the weights W to focus the robot motion on the important part of $\dot{\mathcal{P}}$. This is done by using a weighted norm in a standard minimization problem

$$\psi(\dot{\mathcal{P}}, W) = \underset{\dot{q}}{\mathrm{argmin}} \|J\dot{q} - \dot{\mathcal{P}}\|^2_W \text{ s.t. } \|\dot{q}\|^2 < \dot{q}^2_{\max,e}. \tag{9}$$

We also need to ensure that the grippers do not move too quickly, so we add the constraint that the robot moves no more than $\dot{q}_{\max,e} > 0$. To solve (9) we use the Gurobi [24] optimizer. We use two different Jacobian approximation methods in our model set; a diminishing rigidity Jacobian, and an adaptive Jacobian, which are described below.

Diminishing Rigidity Jacobian The key assumption used by this method [12] is *diminishing rigidity*: the closer a gripper is to a particular part of the deformable object, the more that part of the object moves in the same way that the gripper does (i.e. more "rigidly"). The further away a given point on the object is, the less rigidly it behaves; the less it moves when the gripper moves. Details of how to construct a diminishing rigidity Jacobian are in Appendix B. This approximation depends on two parameters k_{trans} and k_{rot} which control how the translational and rotational rigidity scales with distance. Small values entail very rigid objects; high values entail very deformable objects.

Adaptive Jacobian A different approach is taken in [13], instead using online estimation to approximate $J(q)$. In this formulation we start with some estimate of the Jacobian $\tilde{J}(0)$ at time $t = 0$ and then use the Broyden update rule [25] to update $\tilde{J}(t)$ at each timestep t

$$\tilde{J}(t) = \tilde{J}(t-1) + \Gamma \frac{\left(\dot{\mathcal{P}}(t) - \tilde{J}(t-1)\dot{q}(t)\right)}{\dot{q}(t)^T \dot{q}(t)} \dot{q}(t)^T \ . \tag{10}$$

This update rule depends on a update rate $\Gamma \in (0, 1]$ which controls how quickly the estimate shifts between timesteps.

6 Experiments and Results

We test our method on three synthetic tests and three deformable object manipulation tasks in simulation. The synthetic tasks show that the principles we use to estimate the coupling between models are reasonable; while the simulated tasks show that our method is effective at performing deformable object manipulation tasks.

6.1 Synthetic Tests

For the synthetic tests, we set up an underactuated system that is representative of manipulating a deformable object with configuration $y \in \mathbb{R}^n$ and control input $\dot{x} \in \mathbb{R}^m$ such that $m < n$ and $\dot{y} = J\dot{x}$. To construct the Jacobian of this system we start with $J = \begin{bmatrix} \mathbf{I}_{m\times m} \\ \mathbf{0}_{(n-m)\times m} \end{bmatrix}$ and add uniform noise drawn from $[-0.1, 0.1]$ to each element of J. The system configuration starts at $\begin{bmatrix} 10 \ldots\ 10 \end{bmatrix}^T$ with the target configuration set to the origin. Error is defined as $\rho(t) = \|y(t)\|$, and the desired direction to move the system at each timestep is $\dot{y}_d(t) = -y(t)$. These tasks have no obstacles or stretching, thus β, λ, and $\dot{q}_{\max,o}$ are unused. Rather than setting the utility noise scale η *a priori*, we use an annealing filter

$$\eta(t+1) = \max(10^{-10}, 0.9\eta(t) + 0.1|r(t+1)|) \ . \tag{11}$$

This enables us to track the changing available reward as the system gets closer to the target. All other parameters are shown in App D.

Table 1. Synthetic trial results showing total regret with standard deviation in brackets for all bandit algorithms for 100 runs of each setup.

# of Models	n	m	UCB1-Normal	KF-MANB	KF-MANDB
10	3	2	4.41 [1.65]	3.62 [1.73]	2.99 [1.40]
60	147	6	5.57 [1.37]	4.89 [1.32]	4.53 [1.42]
60	6075	12	4.21 [0.64]	3.30 [0.56]	2.56 [0.54]

To generate a model for the model set we start with the true Jacobian J and add uniform noise drawn from $[-0.025, 0.025]$ to each element of J. For an individual trial, each bandit algorithm uses the same J and the same model set. Each bandit algorithm receives the same random number stream during a trial, ensuring that a more favourable stream doesn't bias results. We ran one small test using a 3×2 Jacobian with 10 arms in order to yield results that are easily visualised. The second and third tests are representative of the scale of the simulation experiments, using the same number of models and similar sizes of Jacobian as are used in simulation. A single trial consists of 1000 pulls (1000 commanded actions); each test was performed 100 times to generate statistically significant results. Our results in Table 1 show that KF-MANDB clearly performs the best for all three tests.

6.2 Simulation Trials

We now demonstrate the effectiveness of multi-arm bandit techniques on three example tasks, show how to encode those tasks for use in our framework, and discuss experimental results. The first task shows how our method can be applied to a rope, with the goal of winding the rope around a cylinder in the environment. The second and third tasks show the method applied to cloth. In the second task, two grippers manipulate the cloth so that it covers a table. In the third task, we perform a two-stage coverage task, covering portions of two different cylinders. In all three tasks, the alignment error $\rho(\mathcal{P}, \mathcal{T})$ is measured as the sum of the distances between every point in $\mathcal{T}$ and the closest point in $\mathcal{P}$ in meters. Figure 2 shows the target points in red, and the deformable object in green. The video accompanying this paper shows the task executions.

All experiments were conducted in the open-source Bullet simulator [9], with additional wrapper code developed at UC Berkeley. The rope is modeled as a series of 49 small capsules linked together by springs and is 1.225m long. The cloth is modeled as a triangle mesh of size 0.5m $\times$ 0.5m for the table coverage task, and size 0.5m $\times$ 0.625m for the two-stage coverage task. We emphasize that our method does not have access to the model of the deformable object or the simulation parameters. The simulator is used as a "black box" for testing.

We use models generated using the same parameters for all three tasks with a total of 60 models: 49 diminishing rigidity models with rotation and translational deformability values k_{trans} and k_{rot} ranging from 0 to 24 in steps of 4, as well as 11 adaptive Jacobian models with learning rates Γ ranging from 1 to 10^{-10} in multiples of 10. All adaptive Jacobian models are initialized with the

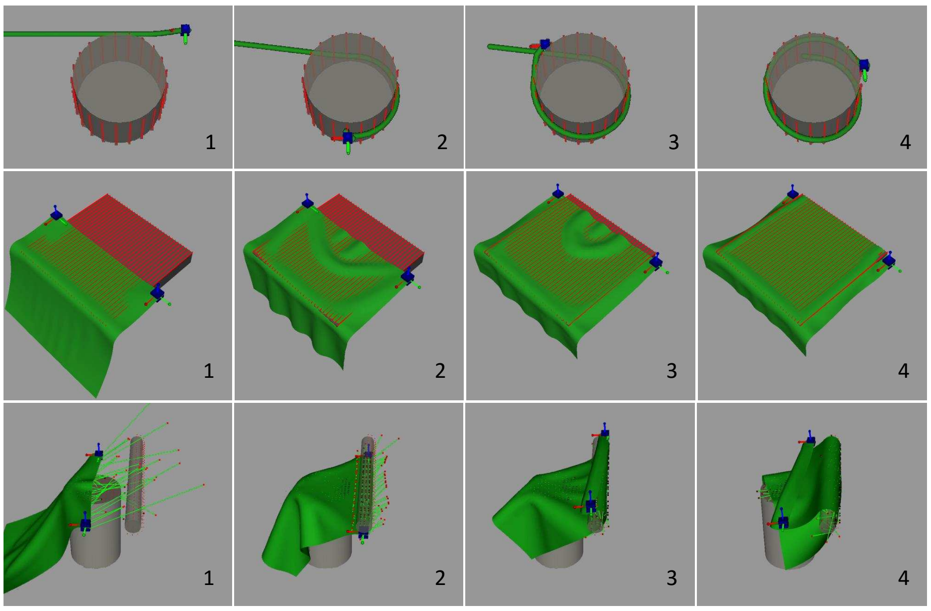

Fig. 2. Sequence of snapshots showing the execution of the simulated experiments using the KF-MANDB algorithm. The rope and cloth are shown in green, the grippers is shown in blue, and the target points are shown in red. The bottom row additionally shows $\dot{\mathcal{P}}_d$ as green rays with red tips.

same starting values; we use the diminishing rigidity Jacobian for this seed with $k_{trans} = k_{rot} = 10$ for the rope experiment and $k_{trans} = k_{rot} = 14$ for the cloth experiments to match the best model found in [12]. We use the same strategy for setting η as we use for the synthetic tests. App D shows all other parameters.

We evaluate results for the MAB algorithms as well as using each of the models in the set for the entire task. To calculate regret for each MAB algorithm, we create copies of the simulator at every timestep and simulate the gripper command, then measure the resulting reward $r_m(t)$ for each model. The reward of the best model $r^*(t)$ is then the maximum of individual rewards. As KF-MANB and KF-MANDB are not deterministic algorithms, each task is performed 10 times for these methods. All tests are run on an Intel Xeon E5-2683 v4 processor with 64 GB of RAM. UCB1-Normal and KF-MANB solve Eq. (9) once per timestep, while KF-MANDB solves it for every model in $\mathcal{M}$. Computation times for each test are shown in their respective sections.

Winding a Rope Around a Cylinder: In the first example task, a single gripper holds a rope that is lying on a table. The task is to wind the rope around a cylinder which is also on the table (see Fig. 2). Our results (Fig. 3) show that at the start of the task all the individual models perform nearly identically, starting to split at 2 seconds (when the gripper first approaches the cylinder) and again at 6 seconds. Despite our model set containing models that are unable to perform the task, our formulation is able to successfully perform the task using all three

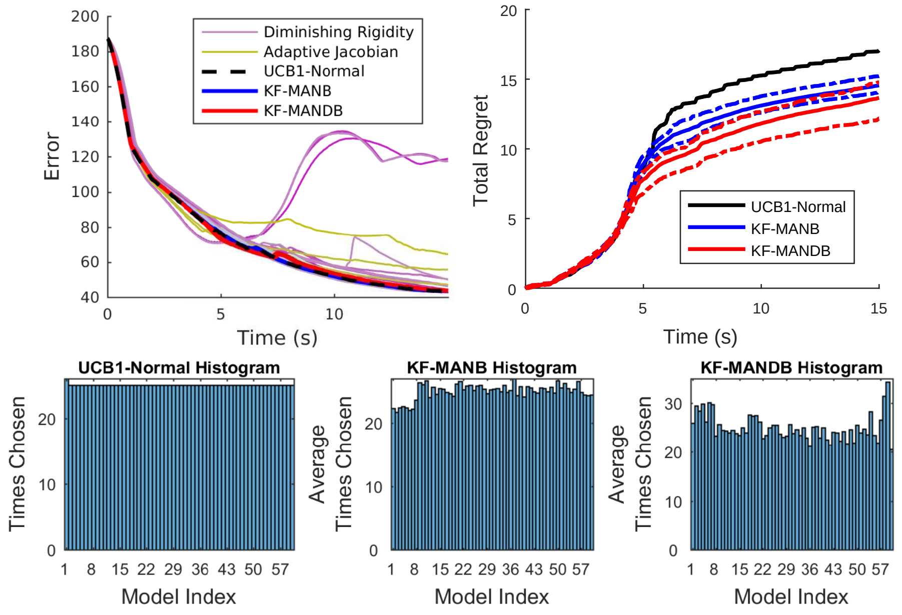

Fig. 3. Experimental results for the rope-winding task. Top left: alignment error for 10 trials for each MAB algorithm, and each model in the model set when used in isolation. UCB1-Normal, KF-MANB, KF-MANDB lines overlap in the figure for all trials. Top right: Total regret averaged across 10 trials for each MAB algorithm with the minimum and maximum drawn in dashed lines. Bottom row: histograms of the number of times each model was selected by each MAB algorithm; UCB1-Normal (bl), KF-MANB (bm), KF-MANDB (br).

bandit algorithms. Interestingly, while KF-MANDB outperforms UCB1-Normal and KF-MANB in terms of regret, all three algorithms produce very similar results. Solving Eq. (9) at each iteration requires an average of 17.3 ms (std. dev. 5.5 ms) for a single model, and 239.5 ms (std. dev. 153.7 ms) for 60 models.

Spreading a Cloth Across a Table: The second scenario we consider is spreading a cloth across a table. In this scenario two grippers hold the rectangular cloth at two corners and the task is to cover the top of the table with the cloth. All of the models are able to perform the task (see Fig. 4), however, many single-model runs are slower than the bandit methods at completing the task, showing the advantage of the bandit methods. When comparing between the bandit methods, both error and total regret indicate no performance difference between the methods. Solving Eq. (9) at each iteration requires an average of 89.5 ms (std. dev. 82.4 ms) for a single model, and 605.1 ms (std. dev. 514.3 ms) for 60 models.

Two-Part Coverage Task: In this experiment, we consider a two-part task. The first part of the task is to cover the top of a cylinder similar to our second scenario. The second part of the task is to cover the far side of a second cylinder. For this task the `GetTargets` function used previously pulls the cloth directly into the second cylinder. The collision avoidance term then negates any motion in that direction causing the grippers to stop moving. To deal with this, we

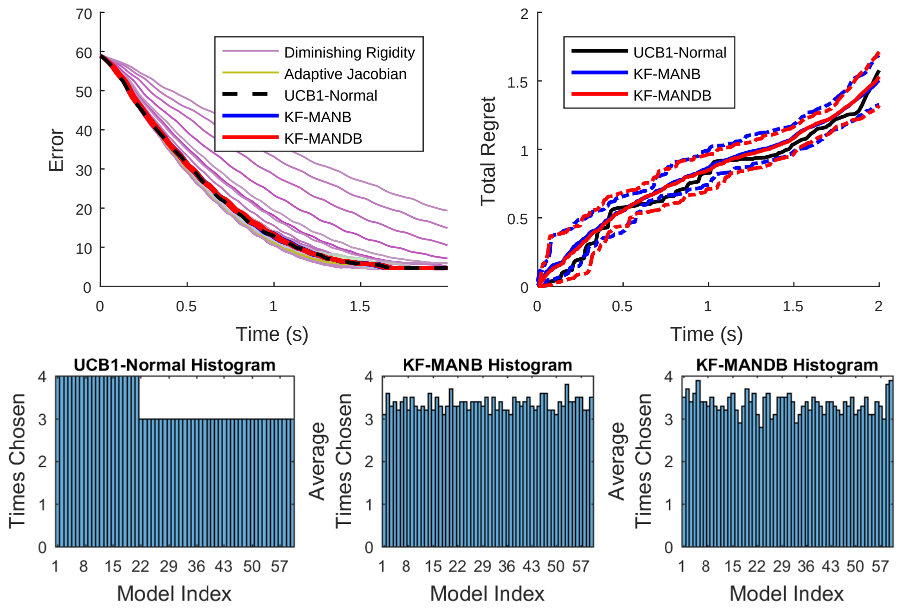

Fig. 4. Experimental results for the table coverage task. See Fig. 3 for description.

discretize the free space using a voxel grid, and then use Dijkstra's algorithm to find a collision free path between each cover point and every point in free space. We use the result from Dijkstra's algorithm to define a vector field that pulls the nearest (as defined by Dijkstra's) deformable object point p_k along the shortest collision free path to the target point. This task is the most complex of the three (see Fig. 5); many models are unable to perform the task at all, becoming stuck early in the task. We also observe that both KF-MANB and KF-MANDB show a preference for some models over others. Two interesting trials using KF-MANDB stand out; in the first the grippers end up on opposite sides of the second cylinder, in this configuration the physics engine has difficulty resolving the scene and allows the cloth to be pulled straight through the second cylinder. In the other trial the cloth is pulled off of the first cylinder, however KF-MANDB is able to recover, moving the cloth back onto the first cylinder. KF-MANDB and UCB1-Normal are able to perform the task significantly faster than KF-MANB, though all MAB methods complete the task using our formulation. Solving Eq. (9) at each iteration requires an average of 102.6 ms (std. dev. 30.6 ms) for a single model, and 565.5 ms (std. dev. 389.8 ms) for 60 models.

7 Conclusion

We have formulated model selection for deformable object manipulation as a MAB problem. Our formulation enables the application of existing MAB algorithms to deformable object manipulation as well as introduces a novel *utility* metric to measure how useful a model is at performing a given task. We have

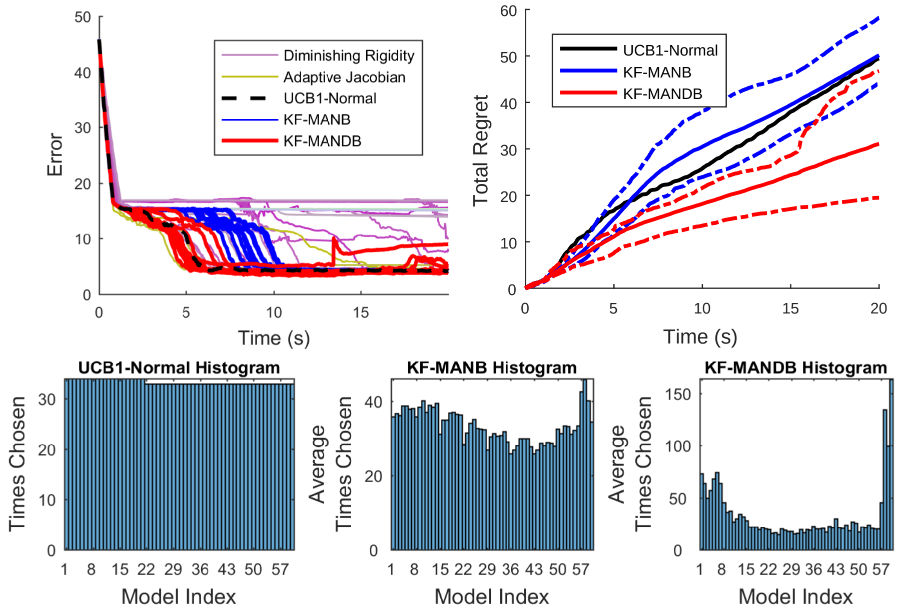

Fig. 5. Experimental results for the two-part coverage task. See Fig. 3 for description.

also presented Kalman Filtering for Non-stationary Multi-arm Normal Dependent Bandits (KF-MANDB) to leverage coupling between dependent bandits to learn more from each arm pull. Our experiments show how to perform several interesting tasks for rope and cloth using our method.

One notable result we observe is that finding and exploiting the best model is less important than avoiding poor models for extended periods of time; in all of the experiments UCB1-Normal never leaves its initial exploration phase, however it is able to successfully perform each task. We believe this is due to many models being able to provide commands that have a positive dot-product with the correct direction of motion.

One limitation of KF-MANDB is handling bifurcations; when very small differences in command sent to the robot cause large differences in the result the assumption of coupling between models in KF-MANDB does not hold. In future work we seek to explore how to overcome this limitation, as well as using the predictive accuracy of each model as an additional measure of model coupling.

8 Acknowledgements

This work was supported in part by NSF grants IIS-1656101 and IIS-1551219. We gratefully acknowledge Calder Phillips-Grafflin for his assistance with Bullet.

References

1. Gibson, S.F.F., Mirtich, B.: A survey of deformable modeling in computer graphics. Technical report, Mitsubishi Electric Research Laboratories (1997)

2. Essahbi, N., Bouzgarrou, B.C., Gogu, G.: Soft Material Modeling for Robotic Manipulation. In: Applied Mechanics and Materials. (April 2012)
3. Maris, B., Botturi, D., Fiorini, P.: Trajectory planning with task constraints in densely filled environments. In: IROS. (2010)
4. Müller, M., Dorsey, J., McMillan, L., Jagnow, R., Cutler, B.: Stable real-time deformations. In: SIGGRAPH. (2002)
5. Bathe, K.J.: Finite Element Procedures. Klaus-Jurgen Bathe (2006)
6. Schulman, J., Ho, J., Lee, C., Abbeel, P.: Learning from demonstrations through the use of non-rigid registration. In: Springer Tracts in Advanced Robotics. Volume 114., Springer International Publishing (2016) 339–354
7. Huang, S.H., Pan, J., Mulcaire, G., Abbeel, P.: Leveraging appearance priors in non-rigid registration, with application to manipulation of deformable objects. In: IROS. (2015)
8. Koval, M.C., King, J.E., Pollard, N.S., Srinivasa, S.S.: Robust trajectory selection for rearrangement planning as a multi-armed bandit problem. In: IROS. (2015)
9. Coumans, E.: Bullet physics library. Open source: bulletphysics.org (2010)
10. Jochen Lang, Pai, D.K., Woodham, R.J.: Acquisition of Elastic Models for Interactive Simulation. IJRR **21**(8) (aug 2002) 713–733
11. Cretu, A.M., Payeur, P., Petriu, E.: Neural Network Mapping and Clustering of Elastic Behavior From Tactile and Range Imaging for Virtualized Reality Applications. IEEE TIM **57**(9) (sep 2008) 1918–1928
12. Berenson, D.: Manipulation of deformable objects without modeling and simulating deformation. In: IROS. (2013)
13. Navarro-Alarcon, D., Romero, J.G.: Visually servoed deformation control by robot manipulators. In: ICRA. (2013)
14. Maron, O., Moore, A.W.: Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation. In: NIPS. (1994)
15. Sparks, E.R., Talwalkar, A., Haas, D., Franklin, M.J., Jordan, M.I., Kraska, T.: Automating model search for large scale machine learning. In: SoCC. (2015)
16. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time Analysis of the Multiarmed Bandit Problem. Machine Learning **47**(2/3) (2002) 235–256
17. Gittins, J., Glazebrook, K., Weber, R.: Multi-armed Bandit Allocation Indices. John Wiley & Sons (2011)
18. Whittle, P.: Restless Bandits: Activity Allocation in a Changing World. Journal of Applied Probability **25** (1988) 287
19. Agrawal, S., Goyal, N.: Analysis of Thompson Sampling for the multi-armed bandit problem. Conference on Learning Theory (2012)
20. Granmo, O.C., Berg, S.: Solving Non-Stationary Bandit Problems by Random Sampling from Sibling Kalman Filters (2010)
21. Pandey, S., Chakrabarti, D., Agarwal, D.: Multi-armed bandit problems with dependent arms. In: ICML, New York, New York, USA (2007)
22. Langford, J., Zhang, T.: The Epoch-Greedy Algorithm for Multi-armed Bandits with Side Information. In: NIPS. (2008)
23. Murray, R.M., Li, Z., Sastry, S.S.: A Mathematical Introduction to Robotic Manipulation. Volume 29. CRC Press (1994)
24. Gurobi: Gurobi optimization library. Proprietary: gurobi.com (2016)
25. Broyden, C.G.: A class of methods for solving nonlinear simultaneous equations. Mathematics of Computation **19**(92) (1965) 577–593

A MAB Algorithm Blocks

A.1 UCB1-Normal

Reproduced from [16].

Loop: For each $n = 1, 2, \ldots$

- If there is a machine which has been played less than $\lceil 8 \log n \rceil$ times then play this machine. If multiple machines qualify, we play the machine that has been played less, selecting the machine with the lower index in the case of a tie.
- Otherwise play machine j that maximizes

$$\bar{x}_j + \sqrt{16 \cdot \frac{q_j - n_j \bar{x}_j^2}{n_j - 1} \cdot \frac{\ln(n-1)}{n_j}} \tag{12}$$

  where $\bar{x}_j$ is the average reward obtained from machine j, q_j is the sum of squared rewards obtained from machine j, and n_j is the number of times machine j has been played so far.
- Update $\bar{x}_j$ and q_j with the obtained reward x_j.

A.2 KF-MANB

Algorithm 6 KF-MANB - reproduced from [20]

```
Input: Number of bandit arms $L$; Observation noise $\sigma_{ob}^2$; Transition noise $\sigma_{tr}^2$.
Initialization: $\mu_q[1] = \mu_2[1] = \cdots = \mu_L[1] = A$; $\sigma_1[1] = \sigma_2[1] = \cdots = \sigma_L[1] = B$; # Typically, $A$ can be set to $0$, with $B$ being sufficiently large
  for $N = 1, 2, \ldots$ do

    1. For each arm $j \in \{1, \ldots, L\}$, draw a value $x_j$ randomly from the associated normal distribution $f(x_j; \mu_j[N], \sigma_j[N])$ with the parameters $(\mu_j[N], \sigma_j[N])$.
    2. Pull the arm $i$ whose drawn $x_i$ is the largest one:
         $i = \operatorname*{argmax}_{j \in \{1,\ldots,L\}} x_j.$
    3. Receive reward $\tilde{r}_i$ from pulling arm $i$, and update parameters as follows:
       – Arm $i$:
         $\mu_i[N+1] = \frac{(\sigma_i^2[N] + \sigma_{tr}^2) \cdot \tilde{r}_i + \sigma_{ob}^2 \cdot \mu_i[N]}{\sigma_i^2[N] + \sigma_{tr}^2 + \sigma_{ob}^2}$
         $\sigma_i^2[N+1] = \frac{(\sigma_i^2[N] + \sigma_{tr}^2)\sigma_{ob}^2}{\sigma_i^2[N] + \sigma_{tr}^2 + \sigma_{ob}^2}$
       – Arm $j \neq i$:
         $\mu_j[N+1] = \mu_j[N]$
         $\sigma_j^2[N+1] = \sigma_j[N] + \sigma_{tr}^2$

  end for
```

B Diminishing Rigidity Jacobian Construction

For every point $p_i \in \mathcal{P}$ and every gripper g we construct a Jacobian $J_{rigid}(q,i,g)$ such that if p_i was rigidly attached to the gripper g then

$$\dot{p}_i = J_{rigid}(q,i,g)\dot{q}_g = \begin{bmatrix} J_{trans}(q,i,g) & J_{rot}(q,i,g) \end{bmatrix} \dot{q}_g \ . \tag{13}$$

Let $D_{i,g}$ be a measure of the distance between gripper g and point p_i. Then the translational rigidity of point p_i with respect to gripper g is defined as

$$w_{trans}(i,g) = e^{-k_{trans}D_{i,g}} \tag{14}$$

and the rotational rigidity is defined as

$$w_{rot}(i,g) = e^{-k_{rot}D_{i,g}}. \tag{15}$$

To construct an approximate Jacobian $\tilde{J}(q)$ for a single point we combine the rigid Jacobians with their respective rigidity values

$$\tilde{J}(q,i,g) = \begin{bmatrix} w_{trans}(i,g)J_{trans}(q,i,g) & w_{rot}(i,g)J_{rot}(q,i,g) \end{bmatrix} \ , \tag{16}$$

and then combine the results into a single matrix

$$\tilde{J}(q) = \begin{bmatrix} \tilde{J}(q,1,1) & \tilde{J}(q,1,2) & \dots & \tilde{J}(q,1,G) \\ \tilde{J}(q,2,1) & \ddots & & \\ \vdots & & & \\ \tilde{J}(q,P,1) & & & \end{bmatrix} \ . \tag{17}$$

C Obstacle Proximity Algorithm

Algorithm 7 Proximity$(g, \mathcal{O})$ - reproduced from [12]

```
1: $d_g \leftarrow \infty$
2: **for** $o \in \{1, 2, \dots, |\mathcal{O}\ \}$ **do**
3:     $p^g, p^o \leftarrow$ ClosestPoints$(g, o)$
4:     $v \leftarrow p^g - p^o$
5:     **if** $\|v\| < d_g$ **then**
6:         $d_g \leftarrow \|v\|$
7:         $\dot{x}_{p^g} \leftarrow \frac{v}{\|v\|}$
8:         $J_{p^g} \leftarrow$ GripperPointJacobian$(g, p^g)$
9:     **end if**
10: **end for**
11: **return** $\{J_{p^g}, x_{p^g}, d_g\}$
```

D Experiment Parameter Values

Table 2. Controller parameters

		Synthetic Trials	Rope Winding	Table Coverage	Two Stage Coverage
$\mathfrak{se}(3)$ inner product constant	c	-	0.0025	0.0025	0.0025
Servoing max gripper velocity	$\dot{q}_{\max,e}$	0.1	0.2	0.2	0.2
Obstacle avoidance max gripper velocity	$\dot{q}_{\max,o}$	-	0.2	0.2	0.2
Obstacle avoidance scale factor	β	-	200	1000	1000
Stretching correction scale factor	λ	-	0.005	0.03	0.03

Table 3. KF-MANB and KF-MANDB parameters

		Synthetic Trials	Rope Winding	Table Coverage	Two Stage Coverage
Correlation strength factor (KF-MANDB only)	ξ	0.9	0.9	0.9	0.9
Transition noise factor	σ^2_{tr}	1	0.1	0.1	0.1
Observation noise factor	σ^2_{obs}	1	0.01	0.01	0.01