Kinodynamic Randomized Rearrangement Planning via Dynamic Transitions Between Statically Stable States

Joshua A. Haustein*, Jennifer King†, Siddhartha S. Srinivasa†, Tamim Asfour*
*Karlsruhe Institute of Technology(KIT) † Carnegie Mellon University(CMU)
joshua.haustein@gmail.com, asfour@kit.edu, {jeking,siddh}@cs.cmu.edu

***Abstract*—In this work we present a fast kinodynamic RRT-planner that uses dynamic nonprehensile actions to rearrange cluttered environments. In contrast to many previous works, the presented planner is not restricted to quasi-static interactions and monotonicity. Instead the results of dynamic robot actions are predicted using a black box physics model. Given a general set of primitive actions and a physics model, the planner randomly explores the configuration space of the environment to find a sequence of actions that transform the environment into some goal configuration.**

In contrast to a naive kinodynamic RRT-planner we show that we can exploit the physical fact that in an environment with friction any object eventually comes to rest. This allows a search on the configuration space rather than the state space, reducing the dimension of the search space by a factor of two without restricting us to non-dynamic interactions.

We compare our algorithm against a naive kinodynamic RRT-planner and show that on a variety of environments we can achieve a higher planning success rate given a restricted time budget for planning.

I. Introduction

We study the *rearrangement planning problem* [6, 9, 11, 17, 21] where a robot must manipulate several objects in clutter to achieve a goal. Rearrangement of clutter can significantly increase the success rate of manipulation tasks [9, 11] and perception [18, 24], and is increasingly significant in unstructured human environments such as homes, workplaces, disaster sites and shared workcells in factories.

Consider a robot clearing a set of toys from a table. As shown in Fig. 1, the table is cluttered with objects. The robot's goal is to move the green box into a crate, weaving through clutter. Early works focused on finding a sequence of pick-and-place actions that moved clutter aside and eventually the target to its goal [22]. Recent work has shown that nonprehensile actions such as pushing, pulling, sliding, and sweeping, enable faster and better solutions, with higher success rates [5, 7, 16].

The problem of rearrangement planning has been shown to be NP-hard [25]. Previous works have used heuristics such as monotonicity to keep the problem tractable [8, 22]. Nonprehensile interactions have been modeled as motion primitives under the quasi-static assumption that any inertial forces can be neglected due to frictional forces [6, 7]. While there exist analytical quasi-static pushing models [10, 12, 15] that allow a fast to compute prediction of the object's behavior during a push, the actions and objects this can be applied to are limited.

We are interested in overcoming these limitations and present a planner that utilizes dynamic nonprehensile actions to rearrange clutter without being restricted to monotonicity. In our example, after the robot pushes the box into the crate its goal is to put a ball into the crate. In this case the robot must reason about the dynamic behavior of the ball as the ball might fall off the table otherwise.

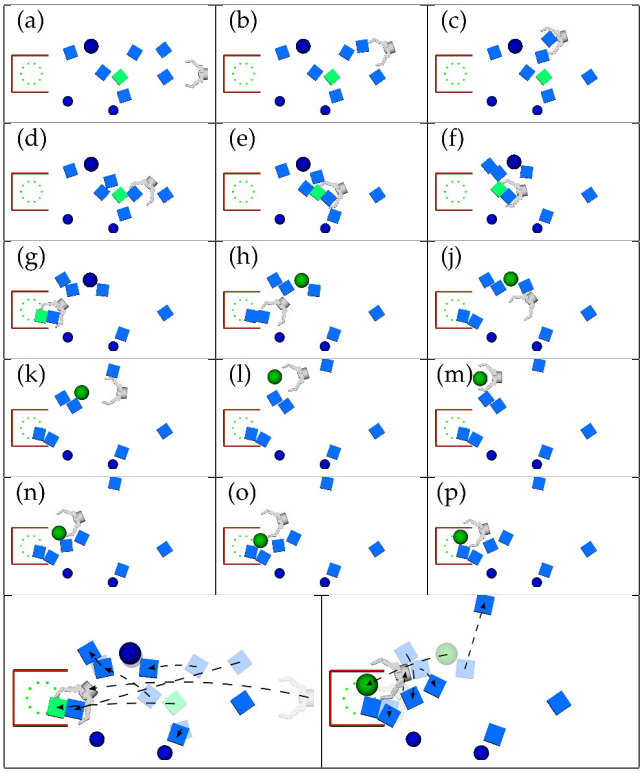
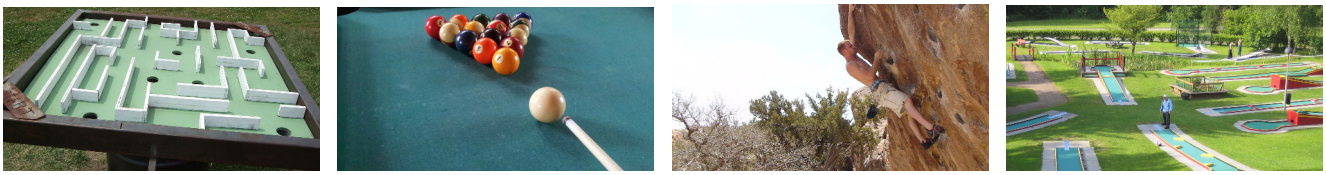


Fig. 1. A robot clearing a table. First, the robot's task is to push the green box to a goal region(green circle) inside of a crate (a)-(g). Afterwards, the robot is supposed to push the green ball to the same area (h)-(p). The bottom row shows the total displacements for each trajectory. The trajectories were generated for a floating end-effector in a $2D$-workspace.

The recent increasing availability of fast dynamic rigid body physics models [1–4] allows modeling dynamic interactions between a robot and the objects in its environment in planning algorithms [27, 28]. Under the quasi-static assumption the rearrangement problem can be formulated as a search for a sequence of robot actions on the joint configuration space of the robot and each movable object in the environment. In contrast, a costly consequence of dynamic interactions is that the search space becomes the full state space of the environment, containing both the configuration and velocity of each object and the robot, doubling the dimension of the search space.



(a) Labyrinth maze[1] (b) Pool billard (c) Bouldering[2] (d) Mini golf[3]

Fig. 2. Examples for dynamic actions between statically stable states utilized by humans.

Our insight is that by choosing dynamic actions that result in statically stable states, e.g. the environment comes to rest after each action, we can search on the lower dimensional configuration space. Consider a bartender sliding a beer bottle to a customer or a child kicking a soccer ball. In both cases, in absence of any external forces other than gravity, the manipulated object eventually comes to rest due to friction. In many activities humans follow a similar approach by transitioning dynamically from one static state into another static state (see Fig. 2).

We contribute by formulating the rearrangement problem as a search for dynamic transitions between statically stable states in the joint configuration space. We show that we can utilize a kinodynamic RRT [14] to solve this search problem. The planner explores the configuration space by randomly sampling dynamic nonprehensile actions and utilizing a dynamic black box physics model to predict the actions' outcomes. To guarantee statically stable states, we require the environment for each action to come to rest after a duration T_{max}. We compare our algorithm to a full-dynamic search and show that with a reasonable choice for T_{max} we can achieve a better planning time even for challenging environments. As a result we show that the probability for our planner to find a solution in a restricted time frame is higher compared to a dynamic planner.

This work is structured in the following way. In Sec. II we formalize the problem of rearrangement planning. Our algorithm is presented in Sec. III and our experiments are described in Sec. IV. In Sec. IV-C we present our results in simulation and on a robot. Finally, we discuss limitations and future work in Sec. V and finish in Sec. VI with a conclusion.

II. The Rearrangement Planning Problem

A. Terminology

In our work an environment E contains a robot R, a target object T, M movable objects the robot is allowed to manipulate and a set $\mathcal{S}$ of static obstacles the robot must not collide with.

We denote the state of movable object $i \in \{1, \dots, M\}$ as $x^i = (q^i, \dot{q}^i) \in \mathcal{X}^i$, where q^i is the object's configuration and $\dot{q}^i$ its velocity. Analogously, we denote the state space of the target object as $\mathcal{X}^T$ and the robot as $\mathcal{X}^R$. We describe the state space of the environment $\mathcal{X}^E$ as the Cartesian product $\mathcal{X}^E = \mathcal{X}^R \times \mathcal{X}^T \times \mathcal{X}^1 \times \ldots \times \mathcal{X}^M$ of the state spaces of all objects in the scene and the robot. We define the free state space $\mathcal{X}^E_{free} \subseteq \mathcal{X}^E$ as the set of states in which no objects are penetrating and the robot is not in collision with any static obstacle in $\mathcal{S}$. Note that this definition specifically allows for movable objects, the target and the robot to be in contact with each other. Furthermore, contact between movable objects or the target and static obstacles is included.

Throughout this work we denote the manifold $\mathcal{C}^E = \{(q, \dot{q}) \in \mathcal{X}^E_{free} \mid \dot{q} = \mathbf{0}\}$ as the environment's statically stable free state space, where every object and the robot are at rest. Note that this is the free configuration space. Likewise, let $\mathcal{C}^T$, $\mathcal{C}^R$ and $\mathcal{C}^i$ denote the statically stable free state spaces for the target, the robot and all movable objects $i \in \{1, ..., M\}$ respectively.

B. Problem formulation

Given a statically stable start state $x_{t_0} \in \mathcal{C}^E$, the goal of the rearrangement problem is to find a sequence of control actions that can be applied to the robot in order to rearrange the environment to a statically stable state $g \in \mathcal{G}$ within a goal region $\mathcal{G} \subset \mathcal{C}^E$. Let $\mathcal{A}$ denote the set of control actions the robot can perform, i.e. joint velocities applied for some duration d. The transition function $\Gamma : \mathcal{X}^E \times \mathcal{A} \to \mathcal{X}^E$ maps a state $x_t \in \mathcal{X}^E$ and an action $a_t \in \mathcal{A}$ at time t to the action's outcome $x_{t'} \in \mathcal{X}^E$ at time $t' > t$, where $d = t' - t$ is the duration of the action. Since we are interested in nonprehensile manipulation, Γ is the laws of physics of the real world. We can describe a solution of the rearrangement problem as a sequence of state-action pairs $\tau = \{(x_{t_0}, a_{t_0}), \dots, (x_{t_i}, a_{t_i}), (x_{t_{i+1}}, a_{t_{i+1}}), \dots, (x_{t_e}, \varnothing)\}$. The sequence starts at the start state x_{t_0} and ends in an end state $x_{t_e} \in \mathcal{G}$. For each two successive pairs (x_{t_i}, a_{t_i}), $(x_{t_{i+1}}, a_{t_{i+1}})$ in the sequence the condition $x_{t_{i+1}} = \Gamma(x_{t_i}, a_{t_i})$ must hold, i.e. each state must be the physical outcome of the previous action applied to the previous state.

C. Complexity

The complexity of the problem arises from three aspects. First, the system is under-actuated. One robot must manipulate several objects. Second, the physics of the manipulation makes the transition function, Γ, non-linear. Third, the dimension of the search space is linear in the number of movable objects in the scene. For example, for rigid objects in a three dimensional workspace and a 7-DOF robot the dimension of the

[1] *Image*: Labyrinth of Failure by Kim Navarre from Brooklyn, NY published on http://commons.wikimedia.org/wiki/File:Labyrinth_of_Failure.jpg, accessed 2014

[2] *Image*: Hueco Tanks Bouldering by Surfsupusa at English Wikipedia published on http://commons.wikimedia.org/wiki/File:Hueco_Tanks_Bouldering.jpg, accessed 2014

[3] *Image*: Bulltoftaparken, Minigolfbanan by User Jorchr on sv.wikipedia published on http://commons.wikimedia.org/wiki/File:Bulltoftaparken,_Minigolfbanan.jpg, accessed 2014

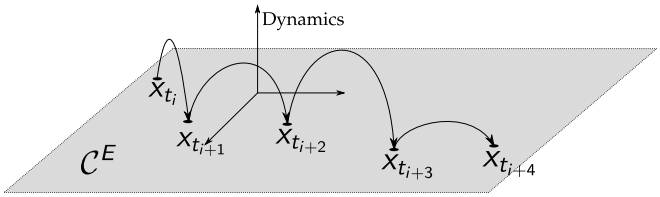


Fig. 3. Dynamic transitions between statically stable states in $\mathcal{C}^E$

search space is $dim(\mathcal{X}^E) = 14 + 12(M+1)$. We reduce the dimensionality of the search space by a factor of two by imposing the restriction $x_t \in \mathcal{C}^E$ for all $x_t \in \tau$. Given the same numbers, the search space's dimension then becomes $dim(\mathcal{C}^E) = 7 + 6(M+1)$. Note that this restriction does not require the environment to be at rest all the time. Instead the environment can behave dynamically during each transition action a_{t_i} from x_{t_i} to $x_{t_{i+1}}$ for any $i = 0..|\tau| - 2$ (see Fig. 3).

III. Forward Search utilizing dynamic black box physics

Randomized sample-based planners have been shown to perform well even on high dimensional state spaces [13, 19]. We follow a similar approach to Zickler et al. [27] and present a kinodynamic RRT algorithm [14] that can solve the rearrangement problem by utilizing nonprehensile motions.

Traditional implementations of the RRT algorithm require a steering method to solve the two-point-boundary problem. The complexity of the transition function Γ and the under-actuation of the system make solving this problem as difficult as solving the full rearrangement problem. Kinodynamic RRTs overcome this limitation.

In Sec. III-A we present the kinodynamic RRT algorithm applied to the rearrangement problem. In Sec. III-B we present the details that are specific to the algorithm when performing a search on the full dynamic state space $\mathcal{X}^E$. In Sec. III-C we present our modification to the algorithm that allows it to search for dynamic transitions between statically stable states in $\mathcal{C}^E$. To distinguish between both variants, we refer to the algorithm presented in Sec. III-B as dynamic planner and to the algorithm presented in Sec. III-C as semi-dynamic planner.

A. Kinodynamic RRT

The underlying kinodynamic RRT algorithm we use is shown in Alg. 1. The algorithm builds a tree with root $x_{t_0} \in \mathcal{X}^E_{free}$ on the state space $\mathcal{X}^E_{free}$ until a goal state $g \in \mathcal{G}$ is added to the tree. The extension of the tree is performed by sampling a state $x_{rand} \in \mathcal{X}^E_{free}$ and extending the tree towards it. For the extension the closest state in the tree x_{near} is picked.

Due to the complexity of Γ we can not trivially compute an action a such that $\Gamma(x_{near}, a) = x_{rand}$. Instead the algorithm uniformly samples k actions $a_i \in \mathcal{A}$ and computes their outcomes x_i in the Propagate function. Each action consists of some control and a duration. In the Propagate function we model the interaction between the robot and any object in the environment by utilizing a dynamic black box physics model $\mathcal{T}$ that approximates the true transition function Γ. The state x_i for $i = 1...k$ that is closest to x_{rand} is then added to the tree. Once a goal state $x_{new} \in \mathcal{G}$ is added to the tree, the search is terminated and the path is extracted from the tree.

The distance function used is a weighted sum of distances on the individual state spaces with weights w_r, w_t, w_m for the robot, target and movable objects:

$$\textsc{Dist}(x, y) = w_r \|x^r - y^r\|_R + w_t \|x^t - y^t\|_T + w_m \sum_{i=1}^{M} \|x^i - y^i\|_M$$

for $x = (x^r, x^t, x^1, \ldots, x^M), y = (y^r, y^t, y^1, \ldots, y^M) \in \mathcal{X}^E$ where $x^r, y^r \in \mathcal{X}^R$, $x^t, y^t \in \mathcal{X}^T$ and $x^i, y^i \in \mathcal{X}^i, i \in \{1, \ldots, M\}$. The norms $\|\cdot\|_R$, $\|\cdot\|_T$ and $\|\cdot\|_M$ are norms on the respective state space, e.g. the Euclidean norm.

The algorithm further requires the specification of a state sampling technique and a propagation method including a validity check. The details about these components for the dynamic and the semi-dynamic planner are presented in the next two sections.

Algorithm 1: The kinodynamic RRT algorithm

Input: The start state x_{t_0}, integer k
Output: A path

```
tree ← {nodes = {x_t0}, edges = ∅}
path ← {}
while not ContainsGoal(tree) do
    x_rand ← SampleConfiguration()
    x_near ← NearestChild(tree, x_rand)
    {a_1, ..., a_k} ← SampleActions(k)
    (x_new, a) ← (∅, ∅)
    for i = 1...k do
        (x_i, a'_i) ← Propagate(x_near, a_i)
        // Dist(∅, x) = ∞ for any x ∈ X^E
        if Dist(x_i, x_rand) < Dist(x_new, x_rand) then
            (x_new, a) ← (x_i, a'_i)
    if (x_new, a) ≠ (∅, ∅) then
        tree.nodes ∪ {x_new}
        tree.edges ∪ {((x_near, x_new), a)}
path ← ExtractPath(tree)
```

B. Dynamic Planner

For the dynamic planner a state x_{rand} is sampled uniformly from $\mathcal{X}^E_{free}$. The Propagate function is presented in Alg. 2. Given an input state $x_{in} \in \mathcal{X}^E_{free}$ and an action a_{in} the physics model $\mathcal{T}$ is utilized to compute the resulting state x_{out}. A resulting state x_{out} is valid if and only if $x_{out} \in \mathcal{X}^E_{free}$ and the robot did not collide with any static obstacle at any time during the execution of a_{in}. Note that the physics model $\mathcal{T}$ guarantees that any resulting state is physically feasible, e.g. no objects intersect.

C. Semi-dynamic Planner

Since the semi-dynamic planner is planning on the statically stable state space $\mathcal{C}^E$, any sampled state x_{rand}

Algorithm 2: PROPAGATE: The dynamic propagate function using a dynamic physics model $\mathcal{T}$.

Input: A state $x_{in} \in \mathcal{X}^E_{free}$, an action a_{in}
Output: The tuple (x_{out}, a_{out}) with $x_{out} \in \mathcal{X}^E$ or $x_{out} = \varnothing$

$x_{out} \leftarrow \mathcal{T}(x_{in}, a_{in})$
$a_{out} \leftarrow a_{in}$
if `IsInvalid(`x_{out}`)` **then**
  $x_{out} \leftarrow \varnothing$
  $a_{out} \leftarrow \varnothing$

and resulting state x_{out} of a propagation need to be at rest, i.e. $x_{rand}, x_{out} \in \mathcal{C}^E$. The state sampling thus uniformly samples statically stable states $x_{rand} \in \mathcal{C}^E$.

The PROPAGATE function is shown in Alg. 3. First, we apply the physics model $\mathcal{T}$ to obtain a resulting state $x' \in \mathcal{X}^E$. This state is not guaranteed to be at rest. Next, we repeatedly apply the physics model $\mathcal{T}$ to compute the duration t_w that is needed for the environment to come to rest if the robot remains inactive. If t_w is less than a fixed maximal duration T_{max} and x' is valid, the resulting statically stable state $x_{out} = x'$ and a modified action a_{out} are returned. The returned action a_{out} is the original action a_{in} followed by a period of the robot being inactive for the duration t_w. In case the environment does not come to rest within T_{max} the propagation is considered a failure and no state is returned. Accordingly, a state $x_{out} = x' \in \mathcal{C}^E$ is considered valid if the robot did not collide with any static obstacle during the execution of a_{in}.

Algorithm 3: PROPAGATE: The semi-dynamic propagate function using a dynamic physics model $\mathcal{T}$

Input: A state $x_{in} \in \mathcal{C}^E$, an action a_{in}, two constants $T_{max}, \Delta t$
Output: The tuple (x_{out}, a_{out}) with $x_{out} \in \mathcal{C}^E$ or $x_{out} = \varnothing$

$t_w \leftarrow 0$
$a_w \leftarrow$ be inactive for time Δt
$x' \leftarrow \mathcal{T}(x_{in}, a_{in})$
while $x' \notin \mathcal{C}^E$ & $t_w < T_{max}$ **do**
  $x' \leftarrow \mathcal{T}(x', a_w)$
  $t_w \leftarrow t_w + \Delta$t
if $t_w < T_{max}$ & `IsValid(`x'`)` **then**
  $x_{out} \leftarrow x'$
  $a_{out} \leftarrow$ do a_{in}, then be inactive for time t_w
else
  $x_{out} \leftarrow \varnothing$
  $a_{out} \leftarrow \varnothing$

IV. EXPERIMENTS & RESULTS

We implement both algorithms in C++ by extending the Online Motion Planning Library (OMPL) framework [23]. We choose Box2D [3] as our physics model, which is a fast $2D$ dynamic rigid body simulator developed for $2D$ games. We run experiments both in simulation and on HERB [20], a bimanual manipulation platform developed in the Personal Robotics Lab at Carnegie Mellon University.

Our experiments explore two hypothesis:

H.1 Given a restricted planning time budget, the semi-dynamic planner achieves a higher success rate than the dynamic planner.

H.2 Increasing T_{max} leads to higher success rates for the semi-dynamic planner.

Hypothesis **H.1** is motivated by the fact that the semi-dynamic planner plans on a smaller search space than the dynamic planner. Given a restricted time budget, we expect the semi-dynamic planner to find a solution faster as long as there is a semi-dynamic solution.

Our second hypothesis **H.2** is motivated by the fact that greater T_{max} allow a greater variety of actions. This is because the larger time frame increases the likelihood for the environment to come to rest.

A. Simulation

Due to the nature of the chosen physics model, all experiments are run in a $2D$ workspace. The state space for each movable object and the target object are $\mathcal{X}^T = \mathcal{X}^i = SE(2) \times se(2)$, the set of poses and twists in the plane. The robot is the floating end-effector of HERB, which is assumed to move holonomically in the plane. Therefore $\mathcal{X}^R = SE(2) \times se(2)$.

The action space $\mathcal{A} = se(2) \times [d_{min}, d_{max}]$ is the set of twists, consisting of a translational x, y-velocity and rotational velocity that are applied for some duration $d \in [d_{min}, d_{max}]$. For the semi-dynamic planner each twist is realized with a ramp profile (Fig. 9, *bottom*), which guarantees the robot being at rest after each action. The number of action samples taken is $k = 10$. The norms on the state spaces are weighted Euclidean distances:

$$\|a, b\| = sqrt((x_a - x_b)^2 + (y_a - y_b)^2 + w_\theta(\theta_a - \theta_b)^2 + w_v((\dot{x}_a - \dot{x}_b)^2 + (\dot{y}_a - \dot{y}_b)^2 + w_\theta(\dot{\theta}_a - \dot{\theta}_b)^2))$$

for $a = (x_a, y_a, \theta_a), b = (x_b, y_b, \theta_b) \in SE(2) \times se(2)$. We choose $w_\theta = 0.001$ and $w_v = 0.25$.

We test 12 different scenes with varying degree of clutter and objects. For the movable objects we distinguish between high friction and low friction objects. While high friction objects almost immediately come to rest, low friction objects tend to continue moving for some time after contact. Some of the scenes contain high friction objects only, some low friction objects only and some are mixed. Additionally, some scenes contain static obstacles that need to be avoided by the robot. Movable objects, however, can collide with static obstacles. The goal region $\mathcal{G} \subset \mathcal{C}^E$ is the set of configurations where the target lies within a circular goal region in the workspace. Three example scenes can be seen in Fig. 8, *top*.

We compare the semi-dynamic planner for different choices of $T_{max} = 0s, 1s, 8s, 20s, 60s$ to the dynamic planner. There are several parameters for both planners that need to be specified. First, the action set is parameterized by velocity and duration limits. Second, the weights in the distance function w_t, w_r, w_m need to be chosen. For each planner, we select parameters for each scene such that the highest success rate is achieved.

All planners are run 110 times on each scene. A run is considered a success if the planner finds a solution

within a planning time of 60s. The success rate is the proportion of successful runs.

B. Real World Execution

Algorithm 4: Jacobian pseudo-inverse trajectory lift

Input: An end-effector trajectory $\tau_E = \{(x_{t_0}, a_{t_0}) \dots (x_{t_e}, \varnothing)\}$, a stepsize Δt
Output: A manipulator trajectory $\tau_M = \{(x'_0, a'_0) \dots (x'_{e'}, \varnothing)\}$

```
q ← SampleStartIK (x_{t_0})
for i = 0 ... e − 1 do
    d ← ExtractActionDuration (a_{t_i})
    t ← 0
    while t < d do
        ψ ← ExtractEEFVelocity (a_i, t)
        φ ← J†(q)ψ + SampleNullspace (q)
        a' ← (φ, Δt)
        τ_M ← τ_M ∪ (q, a')
        q ← q + φΔt
        if not ValidConfiguration (q) then
            return {∅}
return τ_M
```

We wish to validate our trajectories can be executed on a robot. For this, we generate trajectories for the 7-DOF manipulator pictured in Fig. 9. To generate trajectories for our manipulator, we first use our planner to generate trajectories for the end-effector of the manipulator moving in the plane. The actions, $(a_{t_0}, \dots, a_{t_{e-1}}) \in se(2) \times [d_{min}, d_{max}]$, in the resulting trajectories describe a sequence of end-effector twists. Given this sequence, we apply a post-processing step to generate a set of joint velocities $(\phi_0 \dots \phi_{e'-1}) \in \mathbb{R}^7$ that achieve the end-effector motion. We use the Jacobian pseudo-inverse to achieve this conversion. Alg. 4 shows the basic algorithm we use.

We initialize the conversion by sampling a full arm configuration from the set of inverse kinematics solutions that place the end-effector in the configuration specified in x_{t_0}, the initial configuration in the end-effector trajectory. During the conversion, we generate a set of intermediate robot configurations $(q_0 \dots q_{e'}) \in \mathbb{R}^7$. Each intermediate configuration is checked to ensure there is no unmodeled collision, e.g. collision between the forearm and a movable object. If we encounter an invalid configuration, an empty path is returned and an outer process restarts the conversion.

Fig. 9 and Fig. 11 show snapshots of trajectories resulting from this process.

C. Results

Fig. 4 shows the success rate of the semi-dynamic planner for $T_{max} = 8s$ and the dynamic planner on all scenes as a function of planning time budget. As the planners are allowed more time more solutions are found and the success rate grows. For all time budgets the semi-dynamic planner outperforms the dynamic planner. This supports our first hypothesis H.1: **Given a restricted time budget the semi-dynamic planner achieves a higher success rate than the dynamic planner.**

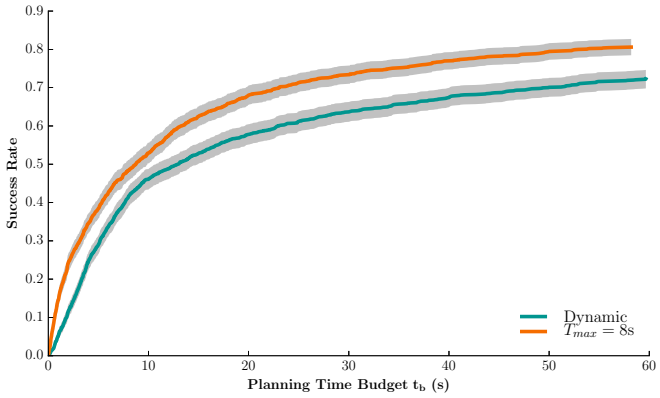


Fig. 4. The success rate of the semi-dynamic planner for $T_{max} = 8s$ and the dynamic planner on all scenes as a function of available planning time budget with 95% Wilson confidence interval [26]. As expected, the dynamic planner performs worse than the semi-dynamic planner on a restricted time budget. Note, however, that given an infinite time budget the dynamic planner is expected to perform at least as well or better.

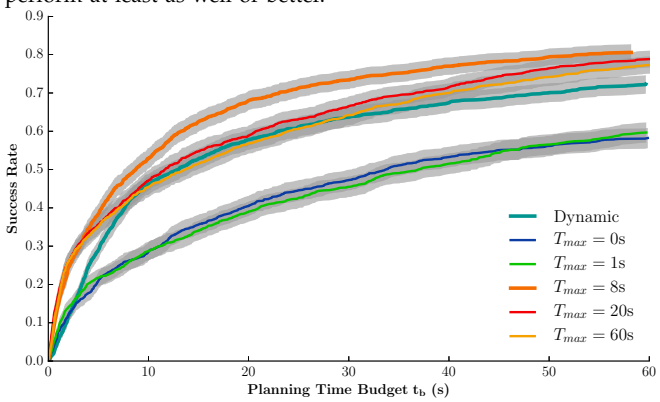


Fig. 5. Success rate of the different planners as function of available planning time budget with 95% Wilson confidence interval. Interestingly, the success rate for $T_{max} = 8s, 20s, 60s$ does not differ largely at $t_b = 60s$. For $T_{max} = 0s, 1s$ the planner performs worst. The success rate can be interpreted as a probability of success.

If we compare the success rates for the different choices of T_{max}, we observe two interesting trends (see Fig. 5). First, the planners with $T_{max} = 0s$ and $T_{max} = 1s$ perform worst. This can be explained by the fact that many scenes require the manipulation of objects that behave dynamically. Consider the scene in Fig. 9. The robot's task is to move the ball into the goal. Finding actions that result in statically stable states after 0s or 1s is unlikely because the ball keeps rolling for some time after contact.

Second, the success rates for $T_{max} = 8s, 20s, 60s$ do not differ largely from each other at a budget of $t_b = 60s$. The planner achieves the highest success rate for any time budget for $T_{max} = 8s$, followed by $T_{max} = 20s$ and then $T_{max} = 60s$. This weakens our second hypothesis H.2 for a restricted time budget: **Increasing T_{max} does not necessarily lead to a higher success rate for a restricted planning time budget.**

While the results support our hypothesis for $T_{max} < 8s$, the benefit of increasing T_{max} vanishes in the range of $[8s, 20s]$. This is surprising since any solution found with some T'_{max} can also be found with

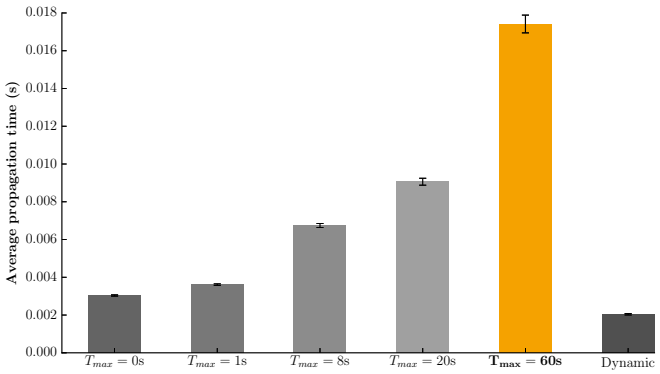


Fig. 6. Average propagation time for a single action. As T_{max} increases the propagation time increases. The propagation time for the dynamic planner is the shortest as it does not require additional time to model the acceleration from rest and deceleration to rest of the manipulator. The error bars show the standard error.

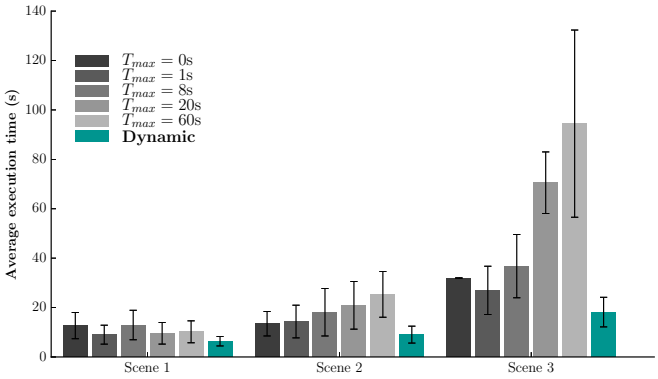


Fig. 7. Average path durations with standard deviation for three different scenes in comparison (also see Fig. 8). Note that there was only a single path found for $T_{max} = 0s$ in Scene 3.

any $T''_{max} > T'_{max}$.

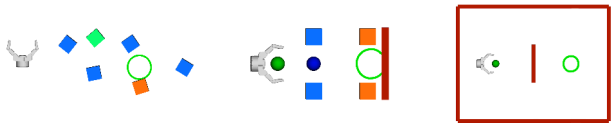
An explanation for this is the average propagation time needed to predict the actions' outcomes, e.g. the time to compute $\mathcal{T}(x, a)$ for some state x and action a. As T_{max} increases more planning time is spent on computing the outcome of actions that result in intermediate states with high velocities and thus need longer to come to rest (see Fig. 6). For smaller T_{max} the propagation of such actions is aborted earlier, saving time for more exploration.

Semi-dynamic example trajectories for a choice of $T_{max} = 8s$ are shown in Fig. 1, Fig. 9 and Fig. 11. In Fig. 1 two subsequent semi-dynamic end-effector trajectories in a cluttered environment are depicted. For both trajectories the task is to move the green object to the goal region marked with a green circle. The robot manipulates multiple objects at once, utilizes object to object collisions and is not restricted in the number of contacts with any object.

The trajectories in Fig. 9 and Fig. 11 are planned in $2D$ and then lifted to a full arm trajectory following the approach presented in Sec. IV-B. In Fig. 9 the robot's task is to move the green ball, which is modeled as a low-friction disc, into the goal on the left side. As can be seen in the velocity profile of the end-effector and the target object, the robot moves the target object from one statically stable state to another. A particularly interesting caging behavior occurs between Fig. 9b, Fig. 9c and Fig. 9e, Fig. 9f. In both cases the robot first accelerates the ball using one side of the end-effector and then catches it at the end of the action using the other side of the end-effector. We observed this behavior frequently in many scenes. An action where the robot pushes a ball into open space is less likely to lead to a statically stable state within T_{max} than an action for which caging occurs.

Fig. 10 shows the velocity profile for a trajectory planned with the dynamic planner on the same scene. Note how both the manipulator and the target object keep in motion for the whole trajectory. As a result the total duration of the trajectory is shorter than the semi-dynamic one. In fact, on average the solutions found by the dynamic planner are shorter in execution time for scenes with dynamically behaving objects.

In Fig. 7 the average execution time is shown on



(a) Scene 1 (b) Scene 2 (c) Scene 3

Type	Scene 1	Scene 2	Scene 3
$Tmax = 0s$	0.98 ± 0.02	0.25 ± 0.08	0.026 ± 0.024
$Tmax = 1s$	0.98 ± 0.02	0.39 ± 0.09	0.10 ± 0.05
$Tmax = 8s$	0.98 ± 0.02	0.98 ± 0.02	0.05 ± 0.04
$Tmax = 20s$	0.98 ± 0.02	0.98 ± 0.02	0.06 ± 0.04
$Tmax = 60s$	0.98 ± 0.02	0.98 ± 0.02	0.30 ± 0.08
Dynamic	0.73 ± 0.08	0.90 ± 0.05	0.27 ± 0.08

Fig. 8. *Top*: Three scenes: the green object is the target object, movable objects are blue and static obstacles are red or orange. Boxes have high friction and balls are modeled as low friction discs. The green circle marks the goal region. *Bottom*: Table showing the 95% Wilson confidence intervals[26] of the success rates for the different planners on the depicted scenes.

three different scenes for each choice of T_{max} and the dynamic planner. In an environment with only high friction objects (Fig. 7, Fig. 8, *left*) the path duration is similar for all planners.

The second environment (Fig. 7, Fig. 8, *middle*) contains two low friction objects, depicted as balls. In this environment the average path duration is higher for each T_{max} than for the dynamic planner. The path duration increases with T_{max}. Note, however, that the success rate on this scene is significantly lower for $T_{max} = 0s, 1s$ than for larger T_{max} (Fig. 8).

The last scene (Fig. 7, Fig. 8, *right*) proves to be difficult for all planners. It contains a static wall blocking direct access to the goal. Furthermore, the goal region lies in open space, where no obstacle prevents the ball from overshooting the goal region. The semi-dynamic planner with $T_{max} = 0s, 1s, 8s, 20s$ found only very few solutions. If a solution was found the path duration is on average shorter than for greater T_{max} as it involves less waiting time. For $T_{max} = 60s$ more solutions are found, but the path duration on average is the longest. The dynamic planner achieved a similar success rate on this environment, but it found paths with the shortest execution times on average.

We run preliminary experiments on a real robot (Fig. 11), which are discussed in the next section.

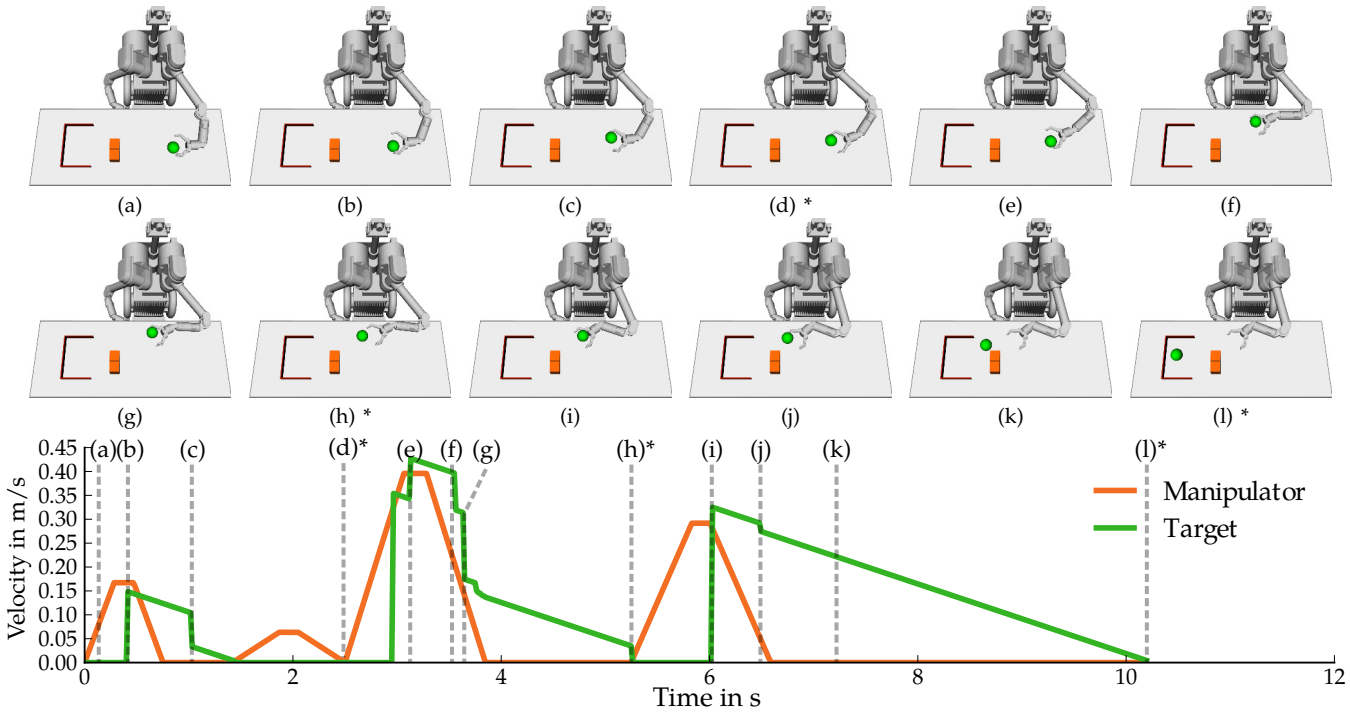


Fig. 9. *Top*: Snapshots from a semi-dynamic trajectory lifted to a 7-DOF arm trajectory and executed on a robot in simulation ($T_{max} = 8s$). The task is to move the green ball into the red goal. The orange boxes are static obstacles. The states marked with * are statically stable. *Bottom*: Velocity profile of the end effector and the target. Note how the robot waits until the target comes to rest before it performs the next action. In order for the robot to reliably come to rest, each action follows a ramp velocity profile.

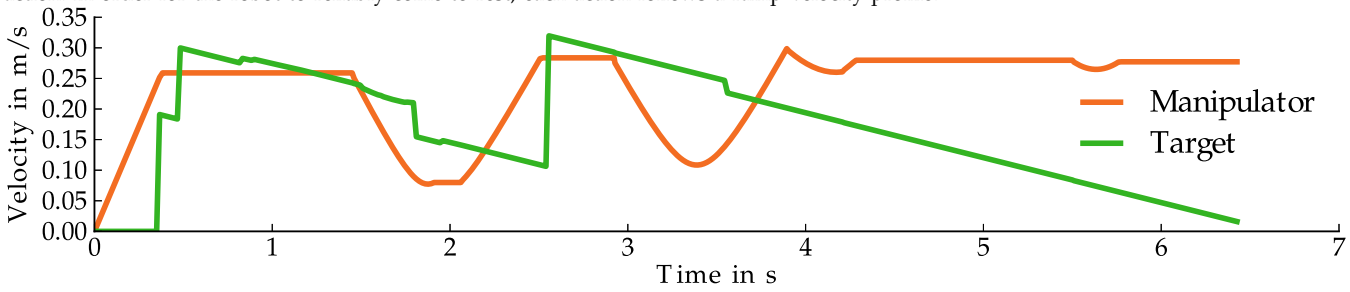


Fig. 10. Velocities for a dynamic trajectory. In contrast to the semi-dynamic trajectory the environment does not come to rest after each action.

V. Limitations & Future Work

Our implementation plans for a floating end-effector in 2*D*-environments and then lifts a trajectory to a full arm trajectory in a post-process. Planning in 2*D* allows us to use the very fast physics model Box2D and restricts the search space to a manifold, where it is most likely for the robot to constructively interact with objects. This approach, however, suffers from two main limitations. First, our implementation does not take the kinematics of the arm into account, which can result in infeasible end-effector trajectories. Second, any contact between the arm and objects is not modeled. As our algorithm is not restricted to 2*D*-environments, future work will incorporate a 3*D* physics model. Note that our current implementation is well applicable to mobile robots operating in a plane.

Despite these limitations, in preliminary experiments we are able to produce executable trajectories on a real robot (Fig. 11). Our physics model is not a perfect reflection of the physical world. The assumption of perfect knowledge of all geometry, mass and friction coefficients is impractical. While we are able to achieve some success executing trajectories open-loop, our experiments highlighted the need to incorporate uncertainty into planning. This is especially the case for rearranging objects with low friction. Here, we believe that the nature of semi-dynamic plans is particularly well suited. While each action requires a dynamic control, there is no time constraint after the execution of an action. The robot waits until the environment comes to rest and then has time to observe the environment and re-plan in case of an execution failure.

VI. Conclusion

We presented a semi-dynamic motion planner that solves rearrangement problems utilizing dynamic non-prehensile actions. In contrast to many previous works our planner does not require monotonicity nor is it limited to quasi-static interactions. We showed that our planner outperforms a dynamic planner in planning time on a variety of scenes. This, however, comes at

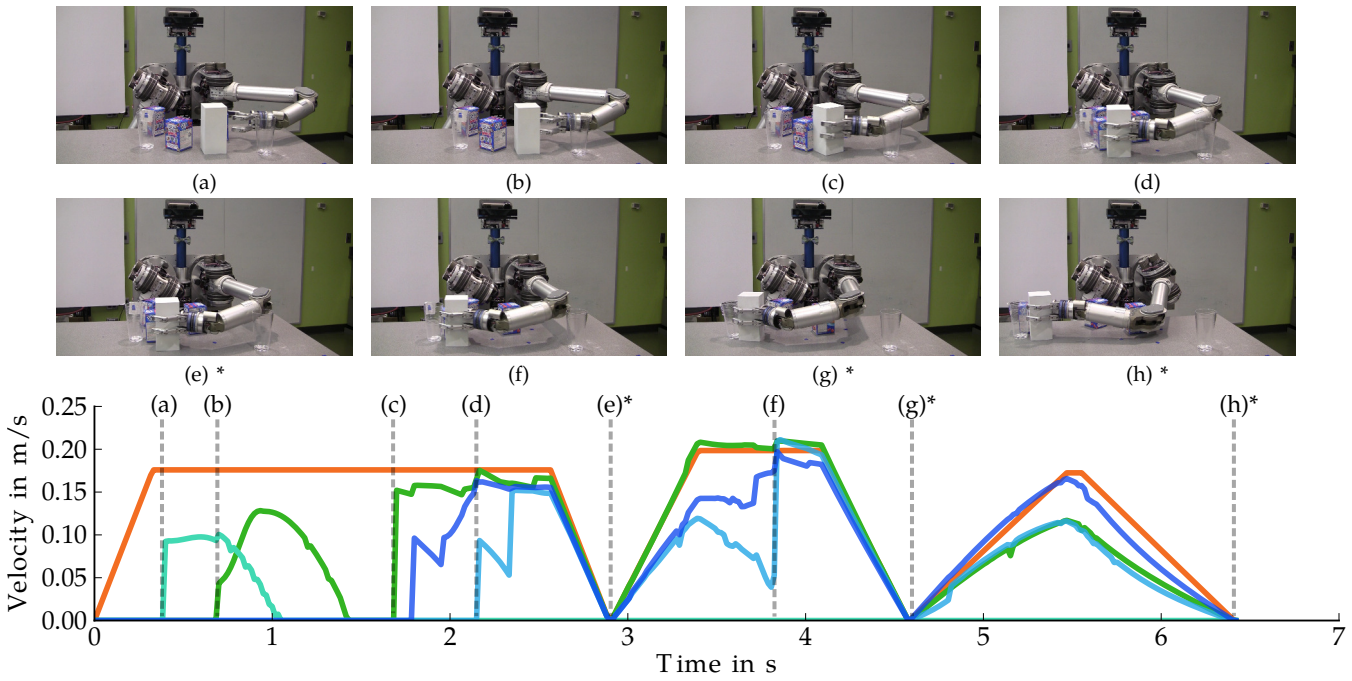


Fig. 11. *Top*: Snapshots from a semi-dynamic trajectory lifted to a 7-DOF arm trajectory and executed on the robot HERB ($T_{max} = 8s$). The task is to move the white box to the left. *Bottom*: Velocity profile for the trajectory as planned. Note how multiple objects are pushed simultaneously. Also the target object, the cup and one of the poptart boxes are moved multiple times. The statically stable states are marked with *.

the cost of an increased execution time in case of the presence of low friction objects. Hence, we highlighted a fundamental trade-off between decreased planning time and increased execution time.

VII. Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement no. 270273 (Xperience), the International Center for Advanced Communication Technologies (InterACT), the IGEL program from the College for Gifted Students of the Department of Informatics at Karlsruhe Institute of Technology and from the Toyota Motor Corporation and NASA NSTRF grant #NNX13AL61H.

References

[1] Open Dynamics Engine. http://www.ode.org, 2000 (accessed August 2014.

[2] NVIDIA PhysX: Physics simulation for developers. http://www.nvidia.com, 2009.

[3] Box2d. http://box2d.org, 2010 (accessed August 2014).

[4] Bullet physics library. http://bulletphysics.org, (accessed August 2014).

[5] J. Barry, K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez. Manipulation with multiple action types. In *ISER*, 2012.

[6] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman. Push planning for object placement on cluttered table surfaces. In *IEEE/RSJ IROS*, 2011.

[7] M. Dogar and S. Srinivasa. A framework for push-grasping in clutter. In *RSS*, 2011.

[8] M. Dogar and S. Srinivasa. A planning framework for non-prehensile manipulation under clutter and uncertainty. *Autonomous Robots*, 2012.

[9] M. R. Dogar, K. Hsiao, M. Ciocarlie, and S. S. Srinivasa. Physics-based grasp planning through clutter. In *RSS*, 2012.

[10] S. Goyal, A. Ruina, and J. Papadopoulos. Planar sliding with dry friction. part 1. limit surface and moment function. *Wear*, 1991.

[11] M. Gupta and G. S. Sukhatme. Using manipulation primitives for brick sorting in clutter. In *IEEE ICRA*, 2012.

[12] R. D. Howe and M. R. Cutkosky. Practical force-motion models for sliding manipulation. *IJRR*, 1996.

[13] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[14] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *IJRR*, 2001.

[15] K. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. In *WAFR*, 1995.

[16] T. Mericli, M. Veloso, and H. L. Akin. Achievable push-manipulation for complex passive mobile objects using past experience. In *AAMAS*, 2013.

[17] D. Nieuwenhuisen, A. Stappen., and M. Overmars. An effective framework for path planning amidst movable obstacles. In *WAFR*, 2008.

[18] D. Schiebener, J. Morimoto, T. Asfour, and A. Ude. Integrating visual perception and manipulation for autonomous learning of object representations. *Adaptive Behavior*, 2013.

[19] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani. Manipulation planning with probabilistic roadmaps. *IJRR*, 2004.

[20] S. Srinivasa, D. Ferguson, C. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. Weghe. HERB: A Home Exploring Robotic Butler. *Autonomous Robots*, 28(1):5–20, 2010.

[21] M. Stilman and J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. In *IEEE-RAS Humanoids*, 2004.

[22] M. Stilman, J. Schamburek, J. Kuffner, and T. Asfour. Manipulation planning among movable obstacles. In *IEEE ICRA*, 2007.

[23] I. Sucan, M. Moll, and L. Kavraki. The Open Motion Planning Library. *IEEE Robotics and Automation Magazine*, 2012.

[24] H. van Hoof, O. Kroemer, and J. Peters. Probabilistic segmentation and targeted exploration of objects in cluttered environments. *IEEE Transactions on Robotics (TRo)*, 2014.

[25] G. Wilfong. Motion planning in the presence of movable obstacles. *Annals of Mathematics and Artificial Intelligence*, 1991.

[26] E. B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 1927.

[27] S. Zickler and M. Velosa. Efficient physics-based planning: sampling search via non-deterministic tactics and skills. In *AAMAS*, 2009.

[28] C. Zito, R. Stolkin, M. Kopicki, and J. L. Wyatt. Two-level RRT planning for robotic push manipulation. In *IEEE/RSJ IROS*, 2012.