Modeling and Planning Manipulation in Dynamic Environments

Philipp S. Schmitt[1], Florian Wirnshofer[1], Kai M. Wurm[1], Georg v. Wichert[1], and Wolfram Burgard[2]

***Abstract*— In this paper we propose a new model for sequential manipulation tasks that also considers robot dynamics and time-variant environments. From this model we automatically derive constraint-based controllers and use them as steering functions in a kinodynamic manipulation planner. The resulting plan is not a trajectory, but a sequence of controllers that react on-line to disturbances. We validated our approach in simulation and on a real robot. In the experiments our approach plans and executes dual-robot manipulation tasks with on-line collision avoidance and reactions to estimates of object poses.**

I. INTRODUCTION

Many automation tasks require robots to manipulate objects. Applications range from industrial assembly to logistics. When tasks or environments change frequently, programming robot motions manually is not viable and robots must generate the necessary motions autonomously. Several aspects of manipulation render this a challenging problem:

1) **Variety of tasks:** Consider picking an object from a moving conveyor belt or manipulating a single large object with two robots. These two tasks put constraints on the motion generation that go beyond collision-free motion planning. To address this variety of tasks it is necessary to employ models that have the expressiveness to capture the constraints of different applications.
2) **Dynamic and uncertain environments:** Estimates of a robot's environment are inherently noisy. Thus, planning motions and executing them open-loop is likely to fail eventually. For this reason, manipulation benefits from instantaneous reactions to measurements such as object poses or detected obstacles.
3) **Sequential interdependence of motions and actions:** Subtle geometric and kinematic differences in manipulation tasks can require entirely different sequences of robot motions, grasps and placements. Consider the example in Fig. 1, where the task of two manipulators is to transport a cube through an opening in a wall. In this example the left robot could directly reach for the cube if the opening was wider.

Aspects 1) and 2) are addressed in the literature on constraint-based task-specification and control [1]. Manipulation planning [2] addresses aspect 3). The recently proposed constraint-graph and the corresponding planner [3] handle aspects 1) and 3). But to the best of our knowledge no previous approach addresses all three aspects simultaneously.

[1] Siemens Corporate Technology, Munich, Germany
[2] Department of Computer Science, University of Freiburg, Germany
The presented research is financed by the TransFit project which is funded by the German Federal Ministry of Economics and Technology (BMWi), grant no. 50RA1701, 50RA1702 and 50RA1703.

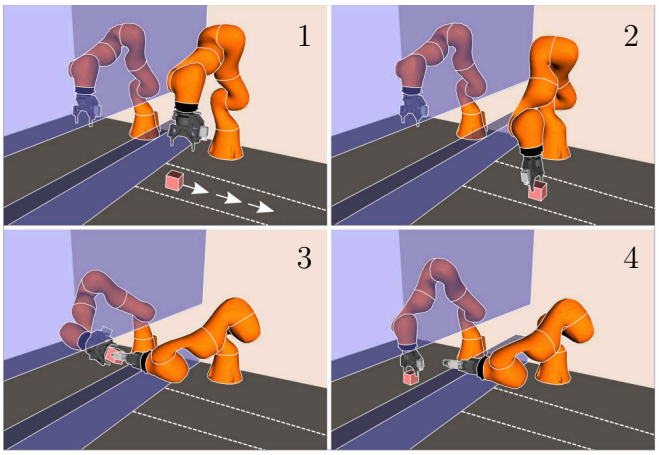

Fig. 1. Example for manipulation in a dynamic environment: The goal of the robot is to place the cube on the left side of the blue wall. The problem is time-variant as the cube is initially moving to the right on a conveyor belt. In order to achieve the goal, the right manipulator must grasp the cube and hand it to the left manipulator through an opening in the wall.

The key idea of this paper is to combine the advantages of constraint-based controllers and manipulation planners that operate on a constraint-graph. The contribution is a new model for planning and controlling manipulation: the dynamic constraint-graph. This model builds on the constraint-graph [3] and extends it to incorporate second-order dynamics as well as time-variance. From this model, we automatically derive constraint-based controllers and use them as steering functions in a kinodynamic manipulation planner. The resulting plan is not a trajectory, but a sequence of controllers that enables on-line reaction to disturbances.

II. RELATED WORK

In the literature on constraint-based task specification and control [1], tasks are modeled as constraints between features of a kinematic tree. From this highly expressive model controllers and state-estimators are derived automatically. Extensions have been proposed, such as the inclusion of inequality constraints [4]. In [5] the eTaSL/eTC framework is presented, that allows the specification of constraint controllers based on expression graphs and supports collision-avoidance based on convex primitives. Manipulation requires not a single robot motion but a sequence of motions and interactions with objects. In [6] a scheduler is proposed that automatically composes and sequences constraint-based controllers based on a constraint satisfaction problem. However, constraint-based controllers are not suitable to address the sequential interdependence of motions and actions that arise in multi-robot manipulation. Our approach addresses this by simulating constraint-based controllers inside a planner.

Reasoning about the sequence of motions of robots and objects in an integrated fashion is known as manipulation planning [2]. Robotic manipulation involves motion through high-dimensional configurations spaces. Sampling-based planners such as probabilistic roadmaps [7] have shown good empirical performance in these configuration spaces and are employed for manipulation planning in [8]. Probabilistically complete algorithms for a broader class of planning problems, called multi-modal motion planning, have been proposed in [9] and [10]. By adapting informed search strategies, originally developed for task planning, the algorithms in [11] and [12] are capable of solving large scale manipulation tasks. Asymptotically optimal manipulation planners were proposed in [13] and [14]. Dedicated manipulation planners rely on problem-specific sub-algorithms such as steering functions or samplers that enable to explore the constrained configuration spaces that arise in manipulation. These sub-algorithms render the modification or extension of planning domains difficult.

An alternative approach is to model manipulation planning as an instance of constrained motion planning, for which [15] provides a survey. In [3] the constraint-graph is introduced as a generic model for manipulation planning domains. This model has the expressiveness to model a variety of sequential manipulation tasks. The HPP framework [16] supports the specification of such constraint-based planning domains and [17] introduces methods to leverage explicit constraints to speed up computations. However, the output of a planner that operates on the constraint-graph is a geometric path for robots and objects and as such is not suitable to react to unforeseen disturbances. Our approach follows the idea of modeling manipulation as constrained motion, but represents plans as a sequence of constraint-based controllers and thus facilitates a reactive execution.

A general approach to combine deliberate planning and reactive execution is to combine local controllers, or funnels, for which a region of attraction is known as a tree that covers the state space [18]. For a one-dimensional manipulation problem, [19] combines a set controllers that are switched in a provably correct way to manipulate two objects. In [20] this approach is experimentally validated for a two-dimensional environment with multiple objects. By combining an optimal manipulation planner with reactive controllers a provably correct feedback manipulation plan can be constructed for two-dimensional environments with known, non-convex and unknown, convex obstacles [21]. These approaches are currently limited to spherical robots in two-dimensional environments. In contrast, our approach scales to scenarios with articulated robots with 14 axes.

III. DYNAMIC CONSTRAINT-GRAPH

We aim to specify manipulation tasks in a modular way that facilitates planning and reactive execution. To this end, this section introduces a new model: the dynamic constraint-graph. This model extends the constraint-graph [3] to incorporate second-order dynamics and time-variance. We will use the setup and the corresponding manipulation task depicted in Fig. 1 as exemplary model instance.

A **configuration** $q \in \mathbb{R}^n$ and its time derivative $\dot{q}$ encode the continuous state of a system. In the dual-arm example this vector q comprises the 14 axis-positions as well as a vector-quaternion representation for the pose of the cube. Time is represented by t.

A **discrete mode** $\sigma \in \Sigma$ encodes the contact-state of a system and is an element of the finite set of modes Σ. In the example the cube may either be at rest on a surface, moved by a conveyor or held by one of two grippers. Since the cube has six sides that may be in contact with surface, conveyor or grippers the example features $|\Sigma| = 24$ discrete modes. **The discrete mode determines both the system dynamics and the constraints that a configuration q must fulfill.**

A system has the control input $u_{\min} \leq u \leq u_{\max}$. In the example u comprises 14 axis-accelerations. Given a mode σ, a system follows a control-affine, second-order dynamic:

$$\ddot{q} = a_\sigma(q, \dot{q}, t) + B_\sigma(q, \dot{q}, t)u, \tag{1}$$

where $a_\sigma(\cdot)$ denotes the acceleration $\ddot{q}$ for zero control input and $B_\sigma(\cdot)$ denotes its affine dependency on u. In the example, the acceleration of the 14 axes is equal to u, independently of σ. The acceleration of the cube depends on the mode σ and the configuration of the robot. In case of a grasp, the acceleration of its pose is determined by the forward-kinematics of the grasping robot. When the cube is placed on a surface its acceleration is zero. If an object is placed on a conveyor belt, equation (1) is time dependent.

Each mode σ is associated with a set of (twice differentiable) constraints f_σ and g_σ, that must hold while the system is in that mode or to transition into that mode:

$$f_\sigma(q, t) = \underline{0}, \qquad g_\sigma(q, t) \leq \underline{0}. \tag{2}$$

In the example the inequality constraints g_σ comprise axis-limits as well as collision avoidance. The geometry of the robots and cube is approximated by convex primitives and their pairwise penetration depth must remain negative. Equality constraints f_σ arise due to the contact-state of the object. If the cube is placed, its center must be at a fixed height above the surface and its contacting side be anti-parallel to the surface. This also implies that the pose of the cube is under-constrained: two translational axes and one rotational axis are free. Additionally, constraints may be imposed at the velocity level:

$$v_\sigma(q, \dot{q}, t) = \underline{0}, \qquad w_\sigma(q, \dot{q}, t) \leq \underline{0}. \tag{3}$$

Axis-velocity-limits are given as inequality constraints. The relative velocity of the cube to the surface or a gripper must be zero when placed or grasped respectively. While the system is within mode σ the constraints f_σ, g_σ, v_σ, and w_σ must be fulfilled. In the remainder of this paper we will omit the velocity constraints v_σ and w_σ for brevity.

Following the ideas of [3] the modes $\sigma \in \Sigma$ form a graph. To transition from a mode σ_1 in this graph to mode σ_2 the second mode must be in the first mode's **set**

of neighbors: $\sigma_2 \in \mathcal{N}(\sigma_1)$. Additionally, the constraints f_{σ_2} and g_{σ_2} of σ_2 must be fulfilled. E.g. to grasp a resting cube the gripper must be positioned in a grasping pose relative to the cube and all constraints on axis-limits and distances of collision bodies must be fulfilled. Switching a mode, i.e. opening or closing grippers in the example, is assumed to happen instantaneously and is an explicit decision of the robot and not part of the system dynamics.

The graph described by the set of modes Σ, the neighborhood-function $\mathcal{N}(\cdot)$ and the corresponding constraints can be constructed automatically as shown in [3]. Some of the constraints of (2) and (3) of the current mode are automatically fulfilled by the system dynamics (1) similar to the explicit constraints of [17]. E.g. while an object is grasped the constraints on the relative velocity of object and gripper are automatically fulfilled by the forward kinematics.

Let $s = (\sigma, q, \dot{q}, t)$ be the **full state** of a system. The goal for a planner is to find a finite sequence of controllers $\{\pi_i\}_{1 \le i \le k}$ that steer the system from a start state s_{start} to a goal region $\mathcal{S}_{\text{goal}}$. The i-th controller computes the control input $u = \pi_i(s)$ based on the current state s. Furthermore, it may terminate and switch to the next controller π_{i+1} or stop execution in case of the last controller. A controller that decides to switch the mode is always terminated. At all times the constraints of the current mode σ and on mode-switching must be fulfilled. An important aspect of our model is that all restrictions on valid solutions are encoded in the constraint functions. No additional validity checks, typically used for collision checking, are part of the planning domain. Fig. 2 shows a valid motion throughout the dynamic constraint-graph. As the finite number of controllers π_i in a plan is processed in a linear sequence, no chattering of the system can occur due to switching of modes.

IV. PLANNING AND CONTROL

The proposed planner operates by transforming the dynamic constraint-graph into task specifications for constraint-based controllers. These controllers can then be simulated during planning as steering functions. A kinodynamic planner, similar to the Expansive Space Tree (EST) [22], uses these steering functions to build a search tree.

A. Kinodynamic Manipulation Planner

Our planner uses two types of controllers as steering functions. Procedure **randomController**$(\sigma) \rightarrow \pi$ returns a controller π that steers towards a randomly chosen robot configuration on a trajectory that satisfies the constraints of mode σ. The procedure **simulateController**$(s, \pi) \rightarrow s_{\text{new}}$ takes a state s and controller π as input and returns the state s_{new} to which the system is moved by the controller. Typically, there is zero probability of randomly steering into the intersection of the constraint-manifolds of two modes. Procedure **transitionController**$(\sigma, \sigma_{\text{new}}) \rightarrow \pi$ takes modes σ and σ_{new} as input and returns a controller π that steers towards the intersection of both mode's constraint-manifolds while satisfying the constraints of σ. Designing

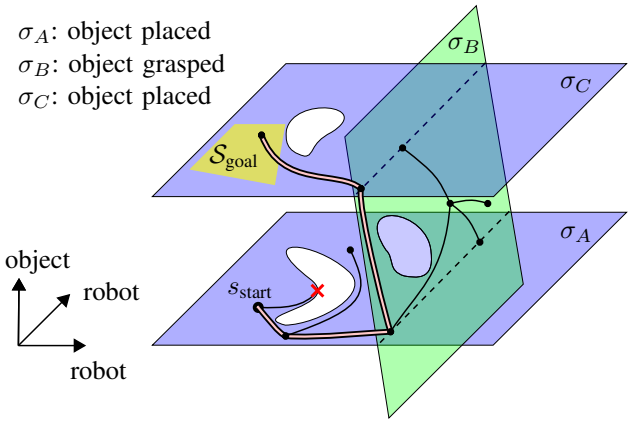

Fig. 2. Model for a pick and place task: The three planes depict the constraint-manifolds of different modes. White areas represent configurations that are in collision. Initially the object is placed (mode σ_A) and the object does not move when the robot moves. At the intersection of modes σ_A and σ_B the object can be grasped. When grasped (mode σ_B) the object is moved along with the robot. Within mode σ_C the object is placed again in a different orientation. The thick line represents a trajectory for robot and object that leads from a start state s_{start} to a goal region $\mathcal{S}_{\text{goal}}$. The thin black lines show alternative motions attempted by the planner.

such controllers is challenging as system dynamics, constraints of the current mode and, in the case of **transitionController**, also the constraints of the neighboring mode must be addressed. This design is discussed in Section IV-B.

Procedure: buildSearchTree(s_{start}) *infinite version*

$N \leftarrow \{s_{\text{start}}\}, \quad E \leftarrow \{\}$
while *true* **do**
  $s \leftarrow (\sigma, q, \dot{q}, t) \leftarrow$ sampleWeighted(N)
  $\pi \leftarrow$ randomController(σ)
  $s_{\text{new}} \leftarrow$ simulateController(s, π)
  N.add$(s_{\text{new}}), \quad E$.add$((s, \pi, s_{\text{new}}))$
  for $\sigma' \in \mathcal{N}(\sigma)$ **do**
    $\pi \leftarrow$ transitionController(σ, σ')
    $s_{\text{new}} \leftarrow$ simulateController(s, π)
    N.add$(s_{\text{new}}), \quad E$.add$((s, \pi, s_{\text{new}}))$

With these procedures we can define our planner. Our algorithm builds a search-tree of nodes N and edges E rooted in the initial state s_{start}. The procedure **buildSearchTree** shows the pseudocode for the tree construction without termination condition. In each iteration the planner samples a random node **from the current set** (forward search!) of nodes N using the procedure **sampleWeighted**. This sampling puts larger weight on nodes that lie in sparsely populated areas of the state-space. From this node we attempt to steer towards a random configuration, as well as to all neighboring modes. Naturally, the implemented planner has no infinite loop and the tree construction is stopped if a node within $\mathcal{S}_{\text{goal}}$ is found or a time budget is depleted. Fig. 2 shows the construction of the search tree. If a controller gets stuck locally its result is discarded after a (simulated) timeout. Fig. 2 shows this as the path that ends in a red cross.

B. Constraint-Based Controllers as Steering Functions

In the following we explain the construction of the controller returned by procedure **transitionController**. This controller builds on the eTC controller [5] but is modified to operate in an acceleration resolved manner following the derivation of [23]. Let us assume our system is currently at a valid state $s = (\sigma, q, \dot{q}, t)$ and should transition to a mode $\sigma' \in \mathcal{N}(\sigma)$. Thus, controls u need to be computed that eventually lead to a state fulfilling the constraints $f_{\sigma'}$ and $g_{\sigma'}$ of mode σ'. Until these constraints are fulfilled the constraints f_σ and g_σ of the current mode σ must not be violated. We can achieve this with control inputs u that obtain the following desired dynamics of the constraint-functions:

$$\begin{aligned}
\ddot{f}_\sigma &= -K_p^{f_\sigma} f_\sigma - K_d^{f_\sigma} \dot{f}_\sigma, \\
\ddot{g}_\sigma &\leq -K_p^{g_\sigma} g_\sigma - K_d^{g_\sigma} \dot{g}_\sigma, \\
\ddot{f}_{\sigma'} &= -K_p^{f_{\sigma'}} f_{\sigma'} - K_d^{f_{\sigma'}} \dot{f}_{\sigma'}, \\
\ddot{g}_{\sigma'} &\leq -K_p^{g_{\sigma'}} g_{\sigma'} - K_d^{g_{\sigma'}} \dot{g}_{\sigma'},
\end{aligned}$$

where, with slight abuse of notation, we denote f, $\dot{f}$ and $\ddot{f}$ as the instantaneous value and time-derivatives of f based on the current q, $\dot{q}$, t and u. The diagonal gain matrices K_*^* must be chosen to achieve stable and at least critically damped dynamics. The reasoning here is: if the dynamics of the constraint functions are stable and properly damped, we will not violate the constraints of the current mode σ, as their constraint-functions are already in the equilibrium state at zero or below zero for inequality constraints. Also, the constraints of the target mode σ' will eventually be fulfilled (within a numerical tolerance).

For manipulation tasks with collision avoidance this system of equations is often over- or under-constrained. The eTaSL/eTC framework [5] allows to separate the constraints into safety constraints, for which the target dynamics must always be obtained, and weighted non-safety constraints. The control inputs are then computed via a quadratic program:

$$\begin{aligned}
\underset{x}{\text{minimize}} \quad & x^\top H x \\
\text{subject to} \quad & L_A \leq Ax \leq U_A \\
& L \leq x \leq U.
\end{aligned} \tag{4}$$

The optimization variable $x = [u, \epsilon_{f'}, \epsilon_{g'}]^\top$ comprises the control inputs u and slack variables $\epsilon_{f'}$ and $\epsilon_{g'}$ for all non-safety constraints. As Hessian matrix H a diagonal matrix is used with weights for all control inputs and one for each non-safety constraint. L and U contain the limits on the control input $u_{\min} \leq u \leq u_{\max}$. The entries of the vectors L_A, U_A and the matrix A are derived from the following equations:

$$\ddot{f}_{\sigma,0} + \frac{\partial \ddot{f}_\sigma}{\partial u} u = -K_p^{f_\sigma} f_\sigma - K_d^{f_\sigma} \dot{f}_\sigma,$$

$$\ddot{g}_{\sigma,0} + \frac{\partial \ddot{g}_\sigma}{\partial u} u \leq -K_p^{g_\sigma} g_\sigma - K_d^{g_\sigma} \dot{g}_\sigma,$$

$$\ddot{f}_{\sigma',0} + \frac{\partial \ddot{f}_{\sigma'}}{\partial u} u = -K_p^{f_{\sigma'}} f_{\sigma'} - K_d^{f_{\sigma'}} \dot{f}_{\sigma'} + \epsilon_{f'},$$

$$\ddot{g}_{\sigma',0} + \frac{\partial \ddot{g}_{\sigma'}}{\partial u} u \leq -K_p^{g_{\sigma'}} g_{\sigma'} - K_d^{g_{\sigma'}} \dot{g}_{\sigma'} + \epsilon_{g'},$$

where $\ddot{f}_0$ is the instantaneous, second-order time-derivative of f with $u = 0$. The reasoning is: the target-dynamics of f_σ and g_σ must be fulfilled otherwise we might violate the constraints of the current mode σ (safety constraints). Reaching the neighboring mode σ' is desirable but optional. Therefore, the target dynamics of $f_{\sigma'}$ and $g_{\sigma'}$ receive slack variables $\epsilon_{f'}$ and $\epsilon_{g'}$ in the optimization (non-safety constraints). The optimization returns a control input that achieves the target dynamics of f_σ and g_σ. Remaining degrees of freedom are used for the weighted goals of using little controls u and achieving the target dynamics of $f_{\sigma'}$ and $g_{\sigma'}$.

The procedure **simulateController** simulates the described controller until the constraints of σ' are fulfilled. In this case the resulting state is returned and added to the tree by the planner. In case the optimization problem of equation 4 is infeasible the procedure returns failure and the corresponding state is rejected by the planner. This may happen when the dynamics of safety constraints are in conflict. Controllers may also become stuck locally. To address this the steering functions returns failure after a timeout is reached. The procedure **randomController** operates in a similar way but the constraints $f_{\sigma'}$ and $g_{\sigma'}$ are replaced by random position constraints on the robot's axes.

C. Controlled Execution

The result of our planning algorithm is a sequence of constraint-based controllers. Fig. 3 shows the phase-portrait of such a controller sequence for a pick and place task. Using this sequence of controllers, an accurately modeled and undisturbed system will reach a goal state. If the execution is disturbed the controllers can meaningfully react. This is due to the fact that the parametrization of the controllers is based on the planning domain (safety constraints) and the decisions of the planner (non-safety constraints).

V. IMPLEMENTATION AND EXPERIMENTS

A. Implementation Details

The definition of constraints (2), (3) and system dynamics (1), computation of all required derivatives and the solution of the optimization problem inside the control loop (4) is based on a custom implementation of the eTaSL/eTC framework [5]. All constraints share the same parameters for desired constraint dynamics K_p^* and K_d^*, assuming unit-free constraint functions. As units of measurement for our constraint functions we used seconds for time, radians for joint angles, meters for translations and components of unit quaternions for rotatory Cartesian quantities. We chose elements of K_p^* as 20 and set K_d^* to achieve a damping ratio of 1.1. The weights for the matrix H where set to 0.001 for the control inputs and to 1 for the slack variables. We added weakly weighted (0.001) non-safety constraints to achieve zero axis velocities that prevent unnecessary motions.

Most of the constraints, such as axis limits and collision constraints, are present in all modes of the constraint-graph.

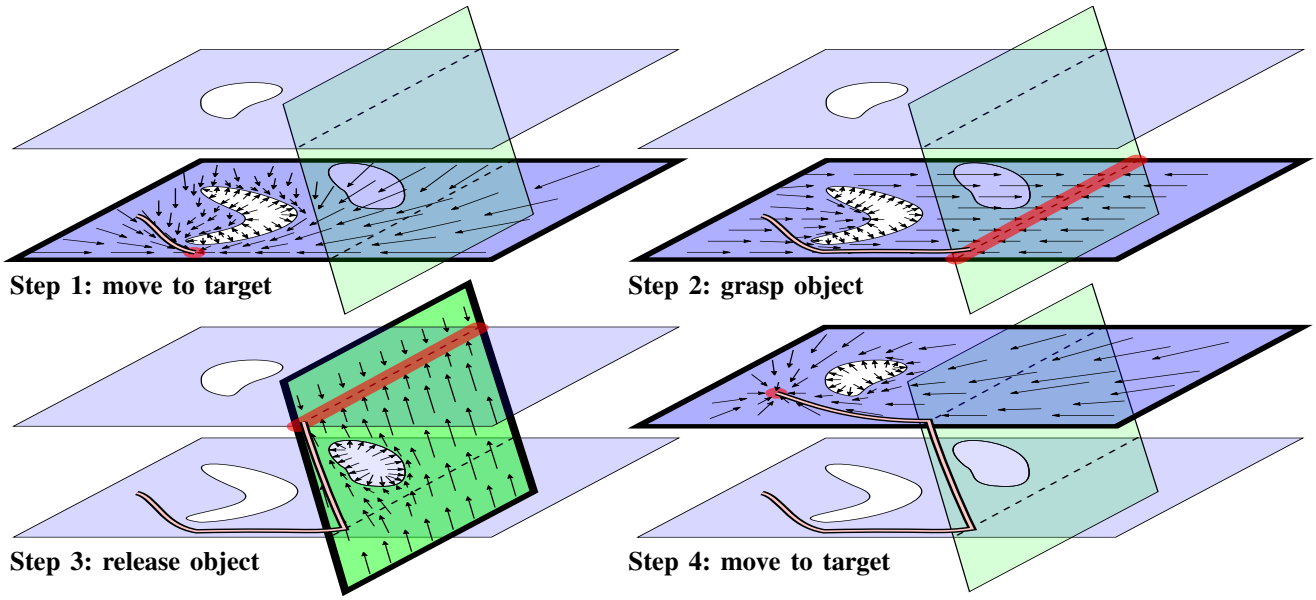

Fig. 3. Phase-portrait of a controller sequence for a pick and place task: In step 1 the goal of the controller is to reach a target axis configuration. This is necessary for step 2, where the goal of the controller is to grasp the object by reaching the intersection of grasp mode and placement mode. If the grasping was attempted first, the system would get stuck in the C-shaped obstacle region. In step 3 the object is placed again at a different placement position. Finally, in step 4, the robot retracts to a home position.

Naturally, these constraints are used only once as safety constraints in the computations. Constraints that are explicitly addressed by the system dynamics of equation (1) are also omitted in the computations.

The time-out parameter for the controller simulation was chosen as 10 (simulated) seconds. As control frequency we use 200 Hz during execution and 40 Hz during simulation to speed up planning. The procedure **sampleWeighted** uses a grid-based discretization of the state space typical for kinodynamic planners. As grid cells we use the mode of the state resulting in 24 cells in the experiments. This is a low number of cells compared to kinodynamic motion planners. However, this is compensated by the constraint-based controllers, that are comparatively powerful (and computationally expensive) steering functions that avoid collisions.

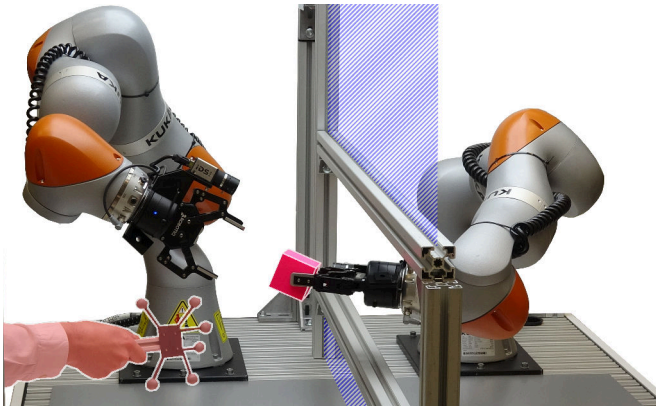

Fig. 4. While the robot is executing a plan, a person, that is tracked via markers, enters the workspace. The person's arm is approximated by a sphere and the robot reacts on-line to satisfy the collision-constraints. This is done while ensuring that no self-collisions occur. Remaining degrees of freedom are used to execute the plan.

B. Experimental Setup

As experimental setup we use the dual-arm robot depicted in Fig. 1. It comprises two seven-axis manipulators, each equipped with a parallel gripper. The goal is to move a cube that is placed on the right to the left side of the setup.

We designed five benchmarks to pose different challenges to manipulation planners. **Benchmark 1** comprises no additional obstacles, i.e. the walls of Fig. 1 are missing and the left manipulator may directly pick and place the cube. **Benchmark 2** adds the lower obstacle and therefore makes a re-grasp necessary, where the right robot hands the object to the left one. **Benchmark 3** and **Benchmark 4** comprise both obstacles with a height of the opening set to 0.4m and 0.2m respectively. These two scenarios where added to stress the motion planning aspects of manipulation. **Benchmark 5** adds time-variance by placing the cube on a conveyor belt as shown in Fig. 1. At the beginning of benchmark 5, the cube is not resting on the surface but moving away from the wall (0.2m opening) with a constant velocity of $5\frac{\text{cm}}{\text{s}}$.

We implemented these benchmark scenarios for the planner presented in this paper as well as for the RMR* algorithm [14]. RMR* is an asymptotically optimal, sampling-based manipulation planner. This allows us to compare both planning-times and costs of the resulting solutions. As the problem setting involves acceleration-controlled robots we implemented RMR* with the trajectory generators of [24] as steering functions. To allow a meaningful comparison of planning times, RMR* was implemented in an incremental fashion that does not separate between roadmap construction and query as presented in [14]. Each benchmark is planned 70 times with different random seeds. The time variant benchmark 5 was not implemented for RMR*. Both planners were implemented in C++, use multiple threads and were run on a ten-core Intel Xeon E5-2650v3.

On the real robot we implemented two qualitative ex-

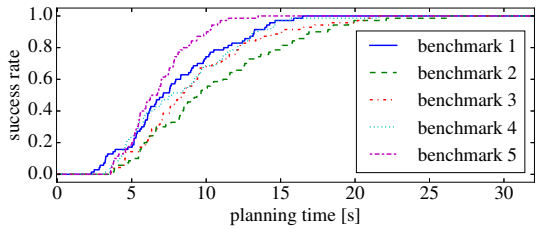

Fig. 5. Success rates of the proposed planner - Each line visualizes the success rate of 70 planning queries for one of the five benchmark tasks.

TABLE I

PLANNING AND EXECUTION TIMES

all values in seconds [std. error of the mean]		Benchmark 1	2	3	4	5
proposed planner	planning time	**7.89** [0.43]	**10.5** [0.59]	**9.27** [0.50]	**8.39** [0.46]	**6.79** [0.26]
	execution time	**14.3** [0.37]	**17.1** [0.36]	**17.7** [0.20]	**17.3** [0.09]	**17.9** [0.14]
RMR*	planning time	**6.44** [0.67]	**8.18** [0.79]	**21.3** [1.54]	**>300** /	**/** /
	execution time of first solution	**10.7** [0.34]	**11.8** [0.44]	**14.9** [0.57]	**/** /	**/** /
	min. time at 60s	**5.76**	**5.90**	**7.19**	**/**	**/**

periments to validate the reactive execution of manipulation plans. The first experiment is designed to incorporate measurements about object poses into the reactive execution. A manipulation plan for benchmark scenario four is planned for an assumed pose of the cube. Then the actual pose of the cube is measured with a camera and the original plan is executed. In the second real-world experiment additional on-line collision-avoidance is added. While a plan is executed we use a tracking-system to detect a person that must be avoided. Fig 4 shows this experiment for benchmark four.

C. Results

The average success rates of our planner over time are shown in Fig. 5. Our planner reliably solves the five benchmark problems. A surprising outcome is the consistency of planning times that appears to be independent of the difficulty of the benchmark. Table I shows the average planning and execution times of plans for both our planner and RMR*. RMR* shows the typical behavior of sampling-based, collision-free planners: planning times deteriorate as more obstacles and "tunnels" are added. Our approach shows relatively consistent planning times across all benchmarks. We attribute this to the nature of the steering functions employed by our approach. As the constraint-based controllers receive the distances between collision-bodies as inequality constraints, collisions simply do not occur, neither during execution nor during planning. This effectively implements a wall-following steering function that allows to efficiently solve the benchmarks with a tunnel in the configuration space (3-5). However, the use of simulated controllers is more time consuming than collision checking and RMR* plans faster when few obstacles are present.

Even though our approach does not attempt to compute optimal solutions, it is interesting to compare the quality of solutions to that of RMR*. RMR* was implemented to minimize the duration of solution trajectories. The first solution returned by RMR* however is arguably representative for typical sampling-based manipulation planners. Table I shows that solutions of our approach have longer durations than both the first solution returned by RMR* and the best solution after one minute of optimization. A possible explanation is that the linear target-dynamic for the constraint-functions under-utilizes the axis-limits on acceleration and velocity compared to the time-optimal local planner of [24]. In practice this means that with our approach the robot moves slower but smoother especially when close to obstacles.

In the real-world experiments the execution of plans reacts to measurements of object poses as well as obstacles that are detected on-line as shown in the accompanying video. The plans consisting of constraint-controllers encode not only the current target state but also the constraints of the planning domain such as axis-limits and self-collision avoidance. Therefore, the robot does not only avoid the detected obstacle during execution, but does so in a way that no self-collisions occur. If redundant degrees of freedom remain, they are used to achieve the current target that is encoded in the plan.

However, it is possible to disturb the robot sufficiently so that the execution becomes stuck locally. This may happen if the controller is blocked by a combination of collision-bodies or by being pushed into a kinematic reconfiguration where axis-limits prevent the controllers to progress towards the goal. In practice this can be easily compensated by stopping and re-planning after a time-out.

A useful property of our approach is that planning and controlled execution both function reasonably in invalid states. Examples include violations of axis-limits as well as objects that are pushed into the collision margin of the robot. The controllers will steer towards valid states according to the target dynamics of the constraint-functions

VI. CONCLUSION

This paper introduced a kinodynamic manipulation planner that operates on a new domain model, called the dynamic constraint-graph. Our planner does not compute a trajectory but a sequence of constraint-based controllers that are derived from the underlying domain model. This allows the robot to react on-line to measurements of object poses or obstacles while executing a plan. As the task specification for the controllers is derived from the planning domain model, safety requirements such as self-collision-avoidance and axis-limits are satisfied during reactive execution. We have successfully validated this approach on a real-world, dual-arm manipulation task with on-line collision-avoidance and reaction to estimates of object poses. Simulated experiments showed that our approach has competitive planning times relative to a (non-reactive) state of the art approach. A current limitation is the lack of formal guarantees regarding probabilistic completeness and regions of attraction for the controller sequence, which is a promising area for future research.

REFERENCES

[1] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *The Int. Journal of Robotics Research*, vol. 26, no. 5, pp. 433–455, 2007.

[2] R. Alami, T. Simeon, and J.-P. Laumond, "A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps," in *The fifth Int. Symposium on Robotics Research*. MIT Press, 1990, pp. 453–463.

[3] J. Mirabel and F. Lamiraux, "Manipulation planning: addressing the crossed foliation issue," in *Int. Conf. on Robotics and Automation*. IEEE, 2017, pp. 4032–4037.

[4] W. Decré, R. Smits, H. Bruyninckx, and J. De Schutter, "Extending iTaSC to support inequality constraints and non-instantaneous task specification," in *Int. Conf. on Robotics and Automation*. IEEE, 2009, pp. 964–971.

[5] E. Aertbeliën and J. De Schutter, "eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs," in *Int. Conf. on Intelligent Robots and Systems*. IEEE, 2014, pp. 1540–1546.

[6] E. Scioni, G. Borghesan, H. Bruyninckx, and M. Bonfè, "Bridging the gap between discrete symbolic planning and optimization-based robot control," in *Int. Conf. on Robotics and Automation*. IEEE, 2015, pp. 5075–5081.

[7] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[8] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *The Int. Journal of Robotics Research*, vol. 23, no. 7-8, pp. 729–746, 2004.

[9] K. Hauser, "Task planning with continuous actions and nondeterministic motion planning queries," in *AAAI Workshop on Bridging the Gap between Task and Motion Planning*, 2010.

[10] K. Hauser and J.-C. Latombe, "Multi-modal motion planning in non-expansive spaces," *The Int. Journal of Robotics Research*, 2009.

[11] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "FFRob: An efficient heuristic for task and motion planning," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 179–195.

[12] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Backward-forward search for manipulation planning," in *Int. Conf. on Intelligent Robots and Systems*. IEEE, 2015, pp. 6366–6373.

[13] W. Vega-Brown and N. Roy, "Asymptotically optimal planning under piecewise-analytic constraints," *The 12th Int. Workshop on the Algorithmic Foundations of Robotics*, 2016.

[14] P. S. Schmitt, W. Neubauer, W. Feiten, K. M. Wurm, G. v. Wichert, and W. Burgard, "Optimal, sampling-based manipulation planning," in *Int. Conf. on Robotics and Automation*. IEEE, 2017, pp. 3426–3432.

[15] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 159–185, 2018.

[16] J. Mirabel, S. Tonneau, P. Fernbach, A.-K. Seppälä, M. Campana, N. Mansard, and F. Lamiraux, "HPP: A new software for constrained motion planning," in *Int. Conf. on Intelligent Robots and Systems*, 2016.

[17] J. Mirabel and F. Lamiraux, "Handling implicit and explicit constraints in manipulation planning," in *Robotics: Science and Systems*, 2018.

[18] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *The Int. Journal of Robotics Research*, vol. 18, no. 6, pp. 534–555, 1999.

[19] H. I. Bozma and D. E. Koditschek, "Assembly as a noncooperative game of its pieces: analysis of 1d sphere assemblies," *Robotica*, vol. 19, no. 1, pp. 93–108, 2001.

[20] C. Karagoz, H. I. Bozma, and D. E. Koditschek, "Feedback-based event-driven parts moving," *Transactions on Robotics*, vol. 20, no. 6, pp. 1012–1018, 2004.

[21] V. Vasilopoulos, T. T. Topping, W. Vega-Brown, N. Roy, and D. E. Koditschek, "Sensor-based reactive execution of symbolic rearrangement plans by a legged mobile manipulator," in *Int. Conf. on Intelligent Robots and Systems.*, 2018.

[22] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *The Int. Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[23] T. De Laet and J. De Schutter, "Constraint-based control of sensor-based robot systems with uncertain geometry," Department of Mechanical Engineering, KU Leuven, Tech. Rep., 2007.

[24] T. Kröger and F. M. Wahl, "Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 94–111, 2010.