2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Macau, China, November 4-8, 2019

Planning Reactive Manipulation in Dynamic Environments

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

***Abstract*— When robots perform manipulation tasks, they need to determine their own movement, as well as how to make and break contact with objects in their environment. Reasoning about the motions of robots and objects simultaneously leads to a constrained planning problem in a high-dimensional state-space. Additionally, when environments change dynamically motions must be computed in real-time.**

To this end, we propose a feedback planner for manipulation. We model manipulation as constrained motion and use this model to automatically derive a set of constraint-based controllers. These controllers are used in a switching-control scheme, where the active controller is chosen by a reinforcement learning agent. Our approach is capable of addressing tasks with second-order dynamics, closed kinematic chains, and time-variant environments. We validated our approach in simulation and on a real, dual-arm robot. Extensive simulation of three distinct robots and tasks show a significant increase in robustness compared to a previous approach.

I. INTRODUCTION

In industrial applications, such as assembly or logistics, many tasks involve manipulating objects. To automate these tasks with robots the positions of objects are typically fixed and the motions of robots are hard-coded for the application. When environments are less structured or even change dynamically this approach is not viable and robots must compute their motion autonomously. Manipulation requires multiple motions of the robot that involve making and breaking contact with the manipulated objects. This sequence of interdependent motions poses a challenging problem for the following reasons:

1) **Variety of tasks and constraints:** Consider opening a door with a mobile manipulator or picking an object from a moving conveyor belt. These exemplary tasks stress the importance of models that can capture the constraints of entirely different applications, e. g., closed kinematic chains and time-variance.
2) **Interdependencies of motions and contacts:** The motion of robots and objects is constrained by contact dynamics. When grasped, an object moves along with a gripper. When placed, it does not move at all. This leads to a problem with discrete and continuous variables, where subtle differences can require entirely different motions. In Fig. 1, the task of a mobile robot is to operate a lever and a valve. If the robot was on the other side of the door this could be done without the additional step of opening the door.

[1] Siemens Corporate Technology, Munich, Germany
[2] Department of Computer Science, University of Freiburg, Germany
The presented research is partially financed by the TransFit project which is funded by the German Federal Ministry of Economics and Technology.

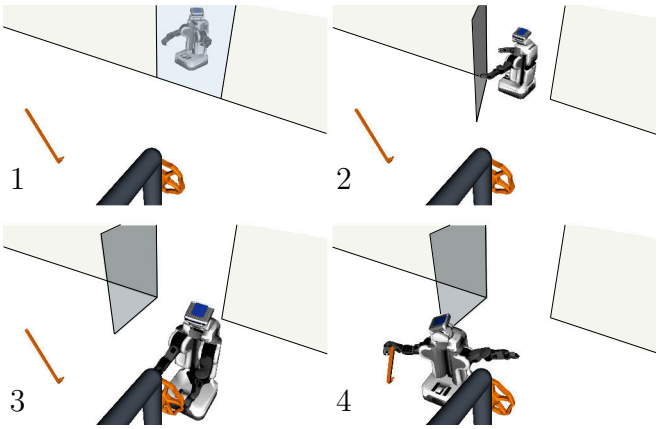

Fig. 1. A manipulation planning problem: A mobile manipulator has to operate a valve and a lever. To do so, a door that blocks the path must be opened first. As additional constraint, the head-mounted camera must point at the lever, door, or valve respectively while they are being manipulated.

3) **Need for real-time reaction:** Robots are dynamic systems and in an increasing number of applications their environments change dynamically as well. Acting in dynamic settings requires closed-loop operation of the robot, i. e., changing action sequences and motions during execution.

Constraint-based task-specification and control [1] addresses the need for real-time control based on generic models for a large variety of tasks. Manipulation planning [2] resolves the interdependencies of motions and contacts along a task. In our previous work [3] we proposed a feedback planner that operates on a constraint-based planning problem and addresses all three challenges. However, the resulting feedback plan is only robust against local deviations of a nominal plan, e. g., it is not possible to react to a part that is accidentally dropped by the robot. Frequent re-planning is not a viable approach since sampling-based planners have a non-deterministic planning time which is problematic for real-time applications.

The contribution of this paper is a feedback planner that computes a global mapping of states to actions for a given manipulation problem. This mapping is then evaluated online in deterministic time to enable a controlled manipulation. Our approach uses a recently proposed, highly general constraint-based model for manipulation tasks [3, 4]. From this model we derive a set of constraint-based controllers that serves as a discrete action-set for a reinforcement learning agent that is trained in simulation. During execution, this agent decides which controller to activate. Extensive simulated experiments validate the increased robustness of our approach for three distinct robots and tasks.

978-1-7281-4003-2/19/$31.00 ©2019 IEEE

II. RELATED WORK

Manipulation requires complex motions and interactions of robots and objects that vary considerably from task to task. Constraint-based task specification and control [1] addresses this by establishing a systematic approach to program such motions. Tasks are specified as constraints on quantities of interest, e. g., sensor readings or Cartesian degrees of freedom of an end-effector. From these constraints controllers are derived automatically. Several extensions to this framework have been developed, such as the inclusion of inequality constraints [5] and geometric path constraints [6]. With the eTaSL/eTC framework [7] the definition of such constraints is done via a graph-structure of computational expressions. This enables to efficiently program complex multi-robot motions with online collision avoidance. However, manipulation requires a sequence of interdependent motions and actions.

This interdependency is addressed with manipulation planning [2], where motions of robots and objects are computed simultaneously. As manipulation requires reasoning about high-dimensional configuration-spaces, sampling based approaches [8]–[10] have shown good empirical performance. Several extensions have been proposed, such as heuristic guidance for large scale planning problems [11, 12] and optimal planners [13, 14].

A shared limitation of these manipulation planners is that it is difficult to adapt them to new domains or to incorporate new constraints to a given domain. This is due to the fact, that these planners rely on problem-specific sampling algorithms and sometimes also steering functions. A more generic approach to planning is to model manipulation as constrained motion, where constraints arise due to contacts [15]. An overview of sampling-based planning methods for such constrained motion can be found in [16]. Mirabel and Lamiraux [4] present the constraint graph as a systematic approach to model different contact states of manipulation and the resulting constraints. Furthermore, a planner is proposed that operates on this model.

The planners discussed so far compute trajectories that solve a planning problem but assume a known and static environment. These trajectories do therefore not encode how to react to unforeseen disturbances. In [17] a feedback planner is presented for one-dimensional manipulation problems and extended to two dimensions in [18]. For two-dimensional manipulation problems and a spherical robot [19] presents a feedback planner that is able to react online to previously unknown obstacles. In our previous work we propose the dynamic constraint-graph [3] as a novel domain model for manipulation planning. This model extends the constraint-graph [4] to second-order dynamics and time-variant systems. From this model, constraint-based controllers are derived and sequenced by a planner. This results in reactive plans that scale to a dual robot scenario with a total of 14 axes. However, the feedback plans of this approach are only locally robust around a nominal trajectory.

A different approach to construct a feedback policy for manipulation is to train a reinforcement learning agent within a physics simulator. Two recent successes can be found in [20] and [21]. Reinforcement learning can capture complex, long-term interdependencies of the decision problem and operates, by construction, in real-time. However, the task specification of reinforcement learning is a reward function, which makes the implementation of hard constraints, like collision-avoidance or precise grasping, difficult.

The approach we present in this paper unifies the strengths of the previously discussed lines of research. As in [3] and [4] we utilize a highly general, constraint-based model to describe the task of a robot. From this model a set of constraint-based controllers is derived, that serve as high-frequency, low-level controllers. The long term reasoning about the sequence of motions and actions is performed by a reinforcement learning agent at a lower frequency. This agent switches, in deterministic time, between the different controllers to achieve its goal.

III. PROBLEM STATEMENT AND NOTATION

The domain model of this paper is a variation of the dynamic constraint-graph [3] with a simplified notation. We use the setup and task depicted in Fig. 1 as a running example to explain this model.

We denote a **configuration** of a system as $q \in \mathbb{R}^n$ and its velocity as $\dot{q}$. In the example of Fig. 1, q comprises 23 values: three for the mobile base, 15 for the torso and the arms, two for the head and one each for the hinges of lever, valve, and door. Time is denoted as t.

In manipulation, the motion of robots and objects is constrained by contacts. A **discrete mode** $\sigma \in \Sigma$ encodes the contact-state of the system and the constraints on configurations. Σ is the finite set of modes. In the example, the door, lever, and valve may be in an open or closed position. Also, at most one of the three may be manipulated by the robot at the same time. Since we predetermine the grippers to be used this results in $|\Sigma| = 20$ modes in the example.

Each mode σ determines **constraints** on configurations via the elements of the vector-valued function f_σ:

$$f_\sigma(q, t) \leq \underline{0}. \tag{1}$$

This may include equality constraints which can be represented as two inequalities with different signs. In the example, when the robot manipulates the lever its gripper must be at a fixed pose relative to the lever. Examples for inequality constraints include limits on axis-positions or distances between geometric bodies for collision-avoidance. The constraints may be time-dependent, e. g., when a conveyor belt is part of the environment.

Additionally, a mode encodes constraints on velocities via the elements of v_σ:

$$\tfrac{\mathrm{d}}{\mathrm{d}t} v_\sigma(q, t) \leq \underline{0}. \tag{2}$$

When a robot grasps an object, the relative velocity of gripper and object must be zero. Maximal velocities of robot-axes form velocity-inequality constraints.

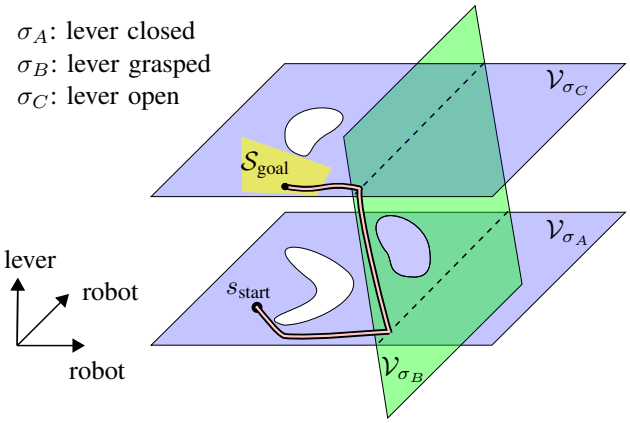

Fig. 2. Model for a manipulation task of moving a lever: Three planes visualize $\mathcal{V}_\sigma$ for different modes. Initially the lever is in a closed position (σ_A) and the robot is not holding it. Thus, the lever does not move regardless of the robot's motion. At the intersection of $\mathcal{V}_{\sigma_A}$ and $\mathcal{V}_{\sigma_B}$ the lever can be grasped. When grasped (mode σ_B) the lever moves along with the robot. Within mode σ_C the lever is released in a different position. The thick line shows a solution trajectory from a start state s_{start} to the goal region $\mathcal{S}_{\text{goal}}$.

The functions f_σ and v_σ are vector-valued, continuous, and piecewise twice-differentiable. As each mode may impose different numbers of constraints we omit the dimensions of the constraint functions for brevity. The modes and constraints can be constructed automatically as shown in [4].

The **full state** of a system $s = (q, \dot{q}, t)$ comprises positions, velocities, and time. We assume that this state is known to the system. For a mode σ we denote the corresponding set of valid states $\mathcal{V}_\sigma$ as the set of states, for which equations (1) and (2) are fulfilled. The complete set of valid states $\mathcal{V}$ is thus the union $\mathcal{V} = \bigcup_{\sigma \in \Sigma} \mathcal{V}_\sigma$.

Given a start state $s_{\text{start}} \in \mathcal{V}$ the goal for a system is now to reach a state within a goal-set $\mathcal{S}_{\text{goal}} \subset \mathcal{V}$ on a continuous, twice time-differentiable path without leaving the set of valid states $\mathcal{V}$. Fig. 2 shows such a motion for manipulating a lever.

As we aim to enable a reactive execution of plans the goal for a planner is to find a controller k of the following form:

$$\ddot{q} = k(q, \dot{q}, t) = k(s), \tag{3}$$

$$\text{with} \quad \ddot{q}_{\min} \leq \ddot{q} \leq \ddot{q}_{\max}. \tag{4}$$

For all start-states $s_{\text{start}} \in \mathcal{V}$ for which a valid motion through $\mathcal{V}$ towards $\mathcal{S}_{\text{goal}}$ exists, this controller must produce such a motion.

Several aspects of our domain model should be noted. First, the output of the controller $\ddot{q}$ contains quantities that are controllable, such as the axis-positions of the robot, and possibly uncontrollable quantities, such as the position of the lever. However, the constraints of equations (1) and (2) force $\ddot{q}$ on physically plausible trajectories. Second, on a physical robot additional actions, such as opening or closing grippers, are required to switch from one mode to another. We assume, that these actions occur automatically and instantaneously and are thus not part of the planning domain. Finally, all constraints on valid motions, **including collision-avoidance**, are part of equations (1), (2), and (4).

IV. FEEDBACK PLANNING FOR MANIPULATION

In this section we explain the proposed feedback planner. To simplify notation we assume that our system keeps track of the current contact-state and thus mode. This mode is written as a function $\sigma(s)$ that takes the current system state s as input. In a scenario as in Fig. 1 this mode is known to a real robot via the state of its grippers.

We address feedback planning for manipulation with a hierarchical control architecture. As in [3] we use the domain model to automatically derive constraint-based controllers, in the following referred to as **low-level controllers**. Two types of low-level controllers are derived: The first type of controller $k_{\sigma'}$ computes a system-acceleration $\ddot{q} = k_{\sigma'}(s)$ that steers the system towards the valid states $\mathcal{V}_{\sigma'}$ of mode σ' without leaving the set of valid states of the current mode $\mathcal{V}_{\sigma(s)}$. This controller is intended to achieve a mode-switch, e. g., make the system grasp or place an object. The second type of controller $k_{q'}$ steers the system towards a given configuration q', again without leaving the valid states of the current mode $\mathcal{V}_{\sigma(s)}$. The purpose of this type of controller is to drive the system around obstacles. Constructing these controllers is a non-trivial task, as they need to consider their target, the second order dynamics, and the possibly time-variant constraints of the current mode. How to construct them will be explained in Section IV-A.

A **high-level controller** chooses which of the low-level controllers is active to control the system. This is done by framing the selection of the active controller as a reinforcement learning problem as described in Section IV-B. Fig. 3 visualizes a phase-portrait of the complete control scheme.

A. Low-Level Controller Synthesis

In the following we explain the construction of the controllers $k_{\sigma'}$ for mode switches and $k_{q'}$ for motions to random targets. The construction of these controllers is identical to that of [3]. We restate this controller-synthesis with the notation of this paper to make this work self-contained. For brevity we omit how the velocity-constraints of equation (2) are integrated into the controller synthesis.

The low-level controllers build on the eTC controller [7]. Our control-architecture switches between different low-level controllers during motion. For this reason it is necessary to use an acceleration-resolved control-scheme similar to [22] to prevent discontinuities in the robot's velocity.

We first explain how the mode-switching controllers $k_{\sigma'}$ operate. Let us assume our system is at a valid state $s = (q, \dot{q}, t)$, within mode σ and should move to the valid states $\mathcal{V}_{\sigma'}$ of mode σ'. We need to compute an acceleration $\ddot{q}$ that steers the system towards $\mathcal{V}_{\sigma'}$ without leaving $\mathcal{V}_\sigma$. This can be achieved with accelerations $\ddot{q}$ that result in the following desired dynamics of the constraint functions f_σ and $f_{\sigma'}$:

$$\ddot{f}_\sigma \leq -K_p^{f_\sigma} f_\sigma - K_d^{f_\sigma} \dot{f}_\sigma,$$

$$\ddot{f}_{\sigma'} \leq -K_p^{f_{\sigma'}} f_{\sigma'} - K_d^{f_{\sigma'}} \dot{f}_{\sigma'},$$

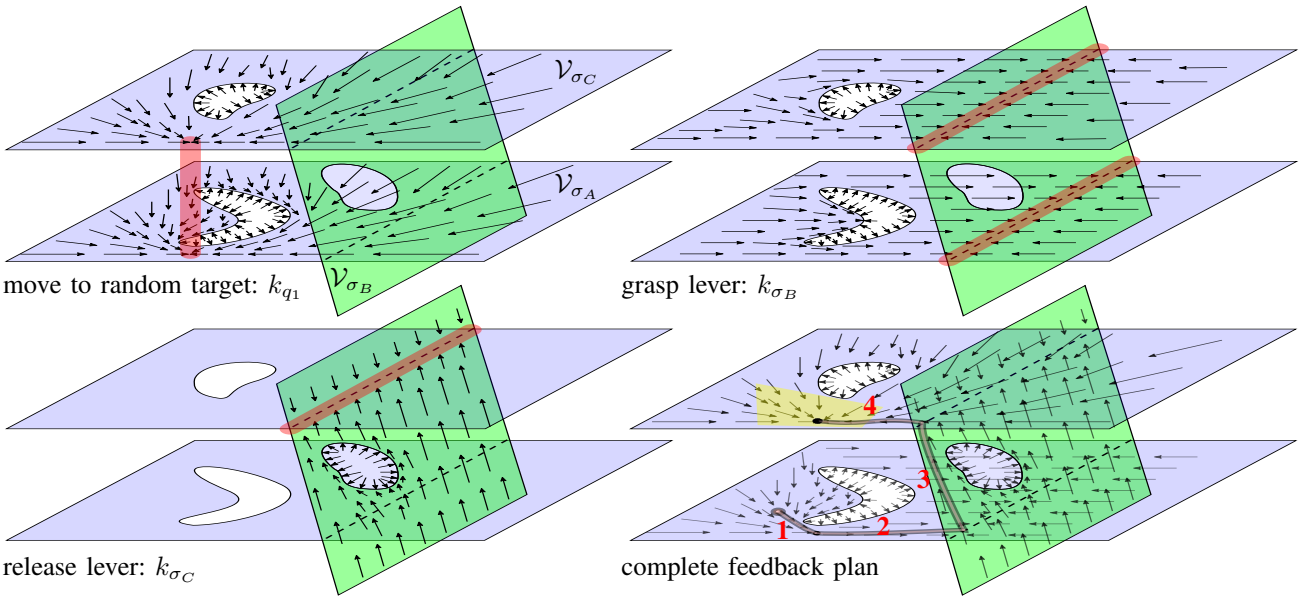

Fig. 3. Constraint-based controllers and construction of the feedback plan (same task of manipulating a lever as in Fig. 2): The upper-left image shows the phase-portrait of a random target controller k_{q_1} that steers to a target configuration q_1 (only drawn for $\mathcal{V}_{\sigma_A}$ and $\mathcal{V}_{\sigma_C}$). In the upper-right and lower-left image two mode-switching controllers k_{σ_B} and k_{σ_C} are shown. The complete feedback plan is obtained by switching between these controllers. This controller is shown in the lower-right image. The thick line visualizes a trajectory of the system towards the goal. For this trajectory a sequence of four active controllers is used: 1: The random target controller k_{q_1} steers around the C-shaped obstacle. 2: k_{σ_B} steers towards a grasp pose. 3: k_{σ_C} is used to move and then release the lever at its new position. 4: k_{q_1} is used again to steer towards the goal region.

with f_*, $\dot{f}_*$, and $\ddot{f}_*$ being the value and time-derivatives of f_* for a given state $(q, \dot{q}, t)$ and acceleration $\ddot{q}$. The matrices K_p^* and K_d^* are diagonal and chosen to achieve stable and at least critically damped dynamics. The reasoning of this approach is: If each vector-element of the constraint-functions follows its own decoupled, stable, and properly damped behavior, eventually all constraints will converge to or stay below zero. As the initial state s is already within $\mathcal{V}_\sigma$ it will stay within $\mathcal{V}_\sigma$.

Typically, the desired dynamics of f_σ and $f_{\sigma'}$ cannot be obtained at the same time. For this reason we relax the dynamics of $f_{\sigma'}$ with slack variables ϵ. This leads to the relaxed, linearized dynamics:

$$\begin{aligned} \ddot{f}_{\sigma,0} + \frac{\partial \ddot{f}_\sigma}{\partial \ddot{q}} \ddot{q} &\leq -K_p^{f_\sigma} f_\sigma - K_d^{f_\sigma} \dot{f}_\sigma, \\ \ddot{f}_{\sigma',0} + \frac{\partial \ddot{f}_{\sigma'}}{\partial \ddot{q}} \ddot{q} &\leq -K_p^{f_{\sigma'}} f_{\sigma'} - K_d^{f_{\sigma'}} \dot{f}_{\sigma'} + \epsilon, \end{aligned} \tag{5}$$

where $\ddot{f}_{*,0}$ denotes the second-order time-derivative of f_* with zero acceleration $\ddot{q} = \underline{0}$. The acceleration $\ddot{q}$ is now computed via a quadratic program:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & x^\top H x \\ \text{subject to} \quad & L_A \leq Ax \leq U_A \\ & L \leq x \leq U. \end{aligned} \tag{6}$$

The optimization variable $x = [\ddot{q}, \epsilon]^\top$ comprises the accelerations $\ddot{q}$ and slack variables ϵ. The matrix H is diagonal with positive weights for the accelerations and the slack variables. L and U contain the limits on acceleration of equation (4). The vectors L_A, U_A and the matrix A are derived from the relaxed, desired dynamics in equation (5).

The result of this optimization ensures that the constraints of the current mode σ will not be violated. Remaining degrees of freedom are used to achieve the target dynamics of $f_{\sigma'}$ while also using as little acceleration as possible.

The construction of the random target controllers $k_{q'}$ proceeds similarly. Given a target configuration q' we replace the constraints $f_{\sigma'}$ with equality constraints $f_{q'}$:

$$q - q' \leq f_{q'}(q, t) \leq q - q'.$$

Other than that the random target controllers operate identically to the mode switching controllers.

B. High-Level Controller Synthesis

The idea behind the high-level controller is to create a finite set of low-level controllers and then switch between them as needed. As the set of modes is finite, there exist only $|\Sigma|$ different mode-switching controllers $k_{\sigma'}$. However, a continuous set of controllers $k_{q'}$ with a configuration q' as target could be constructed. For this reason, we sample $N_r \in \mathbb{N}$ random configurations $\{q_1, ..., q_{N_r}\}$ as targets for N_r controllers. These random configurations do not need to be valid configurations. This leads to a finite set of controllers $\mathcal{K}$ with $|\mathcal{K}| = |\Sigma| + N_r$:

$$\mathcal{K} = \{k_1, ..., k_n\} = \{k_{\sigma_1}, ..., k_{\sigma_{|\Sigma|}}, k_{q_1}, ..., k_{q_{N_r}}\}.$$

The high-level controller operates with a control cycle Δt and should bring the system into a goal state within the shortest possible time. This results in a shortest path problem with a continuous state-space, discrete time-steps (Δt) and a discrete action-set (which controller in $\mathcal{K}$ to activate). The evolution of the time-discrete state $s_i = (q_i, \dot{q}_i, t_i)$ follows the state-transition

function $s_{i+1} = T(s_i, a_i)$, where $a_i \in \{1, ..., |\mathcal{K}|\}$ is the index of the active controller. This state-transition function T simply integrates $\ddot{q} = k_{a_i}(q, \dot{q}, t)$ twice over the time-interval from t_i to $t_{i+1} = t_i + \Delta t$. In case a mode-switching controller was used and the constraints $f_{\sigma'}$ of the target mode σ' are fulfilled within a numerical tolerance, we assume that the tracked mode $\sigma(s)$ switches to σ'.

The goal for the high-level controller is now to find a sequence of actions $\{a_0, a_1, ..., a_{\text{end}}\}$ that leads from the current state to the goal-region $\mathcal{S}_{\text{goal}}$ in a minimum number of steps. It is important to note that the high-level controller is not responsible for any validity checks on the result of applying a low-level controller. All safety requirements, such as collision avoidance, are addressed by the low-level controllers. Even though time and the action-space are discretized this shortest path problem is still challenging due to the continuous and high-dimensional configuration-space. We address this by formulating the problem as a reinforcement learning problem in which each action yields a negative reward of -1 and termination occurs in the goal set $\mathcal{S}_{\text{goal}}$. This means that the rewards of the reinforcement learning problem are sparse. The sparsity of rewards is compensated by the comparatively powerful low-level controllers that enable to effectively explore the constraint-manifolds.

A policy is trained in simulation to form the high-level controller. For this simulation it is sufficient to integrate the accelerations computed by the active controller. Due to the continuous state-space and finite action-space the Deep-Q-Networks (DQN) of [23] are a suitable learning method and were chosen for our approach. We use the full state $s = (q, \dot{q}, t)$ as well as a one-hot encoding of the current mode $\sigma(s)$ as input to this network. The control flow of our architecture can be seen in Fig. 4. As the controllers can be simulated several times faster than required for real-time execution and simultaneously in multiple threads, we use a policy roll-out during execution to further stabilize the policy.

An implication of using only a finite number of random target controllers is that the approach may be incomplete for a planning problem given a choice of random targets. Adapting our method to incorporate policy gradient methods with continuous action-spaces for the high level controller could address this. However, it is important to note that the random targets have a different purpose than, e. g., random samples in a probabilistic roadmap planner [24]. Typically, the system cannot reach a random target within one cycle of the high-level controller, if it can reach it at all. Therefore, the samples serve as directions in configuration-space along which the random target controllers may steer the system. This means that even with a low number of targets N_r, the high-level controller is typically able to move the system on a rich set of trajectories.

Why do we use reinforcement learning to construct the high-level controller? Other methods, such as composing funnels [25] could be used to construct a feedback controller. The reasoning is that these methods require us to estimate a region of attraction for a controller, i. e., to perform computations that essentially go backwards in time. For manipulation

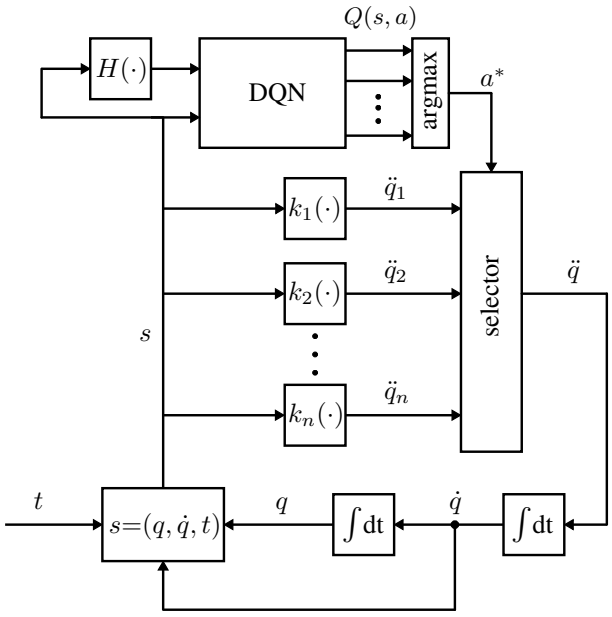

Fig. 4. Control flow of the hierarchical controller: Given a current state s the block $H(s)$ produces a one-hot encoding of the current mode $\sigma(s)$. This encoding as well as the state s itself form the input to a deep-Q-network (DQN) that computes the state-action values $Q(s, a)$ of selecting a controller k_a. The controller with the maximal state-action value is selected and its output is used as the new system acceleration $\ddot{q}$. This acceleration is integrated twice to close the loop.

this is problematic due to the so called crossed foliation issue [4]. For nearly all pairs of valid configurations there exists no connecting path that does not involve at least one mode switch. This makes the computation (or even definition) of regions of attraction difficult. Reinforcement learning bypasses this by only requiring forward simulations. In our previous work [3] we address this by using a kinodynamic planner that also has no need for backwards computations.

V. IMPLEMENTATION AND EXPERIMENTS

A. Implementation Details

a) Low Level Controller: The definition of the constraints (1) and (2), the computation of all required derivatives, and the solution of the quadratic program (6) is based on the eTaSL/eTC framework [7]. The entries of the diagonal matrices K_p^* and K_d^* are chosen equally for all constraints. We chose the entries such that each individual constraint achieves a critically damped behavior with a time-constant of $0.2\,\text{s}$. The low-level controllers operate at a rate of $200\,\text{Hz}$. We set the weights for the matrix H to 0.001 for the accelerations and to 1.0 for the slack variables. To prevent erratic null-space motions we add soft-constraints (weight=0.001) to achieve zero axis-velocities to all controllers. The number of random-target controllers N_r was chosen as 20.

b) High Level Controller: The focus of this paper is not on reinforcement learning but uses it as a tool to create the high-level controller. For this reason, the network architecture and the training were designed to minimize implementation and debugging efforts.

As the ideal depth of the neural network is likely to be different for different tasks, we chose the ResNet [26]

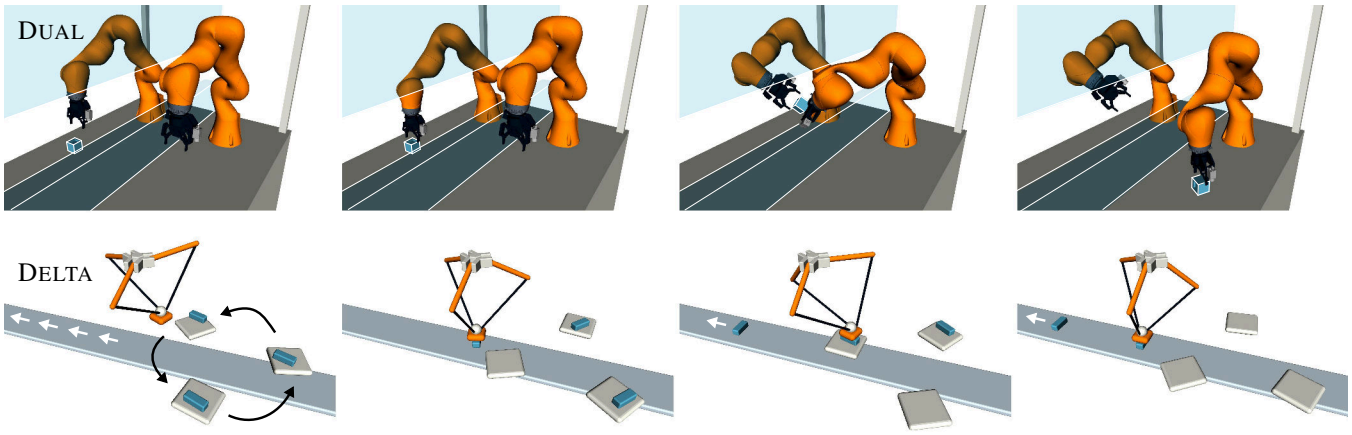

Fig. 5. DUAL-Task: The cube on the left must be placed on the right side of the setup. To do so, the left robot must grasp the cube and hand it to the right robot through a window in a wall. DELTA-Task: A delta-robot must grasp three boxes from rotating carriers and place them on a conveyor belt. Both for grasping and releasing the boxes the robot must synchronize its movement to the time-variant environment.

architecture for the main body of the neural network. This architecture is designed to allow the network to mimic the behavior of more shallow networks if needed. The first layer of our network is a dense layer with 64 units followed by six ResNet bocks, each with two 64 unit dense layers and ELU activations [27]. As the high-level controller has a considerable action-space ($|\Sigma| + N_r = 20 + 20$ actions for the example in Fig. 1) training is suffering from the typical over-estimation of Q-learning. For this reason we employ the combination of the final Dueling-Layer and double Q-learning as presented in [28]. The total number of hidden layers is thus 13.

We strictly separate between data generation and learning. Data generation is done with 10^4 episodes with valid, random start-states and at most 1000 steps of random actions per episode. As a mode-switch requires some time for the controllers to converge, we randomly repeat actions between zero and ten times. The frequency of the high-level controller was chosen as 1 Hz. We use the resulting data set as one large replay buffer without running new episodes during training.

The discount rate was set to $\gamma = 0.95$. We optimize the weights of the network with the Adam optimizer [29] with a learning-rate of 10^{-4}, its parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and a batch size of 320. The weights of the target network are updated after one epoch through the entire data set and we train for a total of 100 epochs. For the policy roll-out we use four actions with the highest Q-values and perform a roll-out of four time steps, i. e., a total of four simulated seconds.

B. Experimental Setup

To evaluate our approach we designed three different manipulation-tasks as benchmarks. All benchmarks involve motion through high-dimensional state-spaces with at least 20 degrees of freedom.

The **DUAL** task requires a dual-arm robot to transport a cube as shown in Fig. 5. It is designed to incorporate the typical interdependencies between motions, grasps, and placements of manipulation planning. The gap in the wall is 30 cm wide and thus forms a narrow tunnel in configuration-space. This poses a difficult motion planning problem as well.

In the **DELTA** task a delta-robot has to pick three boxes from rotating carriers and place them on a conveyor. This benchmark includes closed kinematic chains, multiple objects, and a time-variant environment. To pick or to place the boxes the robot must synchronize its motion with the carriers or the conveyor respectively.

The **PR2** task is shown in Fig. 1 and consists of a PR2 robot that must operate a lever and a valve. To reach lever and valve a door that blocks the way must be opened first. While the PR2 is manipulating either lever, valve, or door its head-mounted camera must point at the manipulated object. This benchmark is created to verify that our approach can address problems other than pick-and-place.

For these benchmarks we implemented the proposed approach as well as the planner of our previous work in [3]. This planner is the only previous approach we are aware of that computes feedback plans for manipulation with articulated robots. We refer to this planner as the "**constraint planner**" in this section. Before the experiments, we generate ten sets of random targets and corresponding simulated training datasets for each benchmark. This is used to train ten DQNs per benchmark. Training multiple DQNs is necessary for the evaluation as the set of random targets, the training dataset, and the initial network weights are random variables.

The experiments are designed to assess the robustness against disturbances in the configuration q and the quality of the computed solutions. To evaluate the robustness of both approaches we proceed as follows: For the constraint planner we randomly sample a valid initial state and compute a plan. Then a second random and valid state within the same mode is sampled and the plan is executed for this disturbed state. It is necessary to sample a state within the same mode since the constraint planner, unlike the proposed approach, cannot react to an unexpected mode change. Each trained agent of our approach is also run on a random, valid initial state (our approach does not need to plan once a policy is trained). As

TABLE I

SUCCESS RATES - DISTURBED EXECUTION

Benchmark		DUAL	DELTA	PR2
proposed approach	max	100%	100%	100%
10 different DQNs	median	99.5%	100%	100%
	mean	98.1%	99.9%	98.5%
	min	93.0%	99.0%	86.0%
constraint planner [3]		11%	97%	57%

each benchmark can be solved in less than 20 seconds, we assume that execution has failed if the goal is not reached within 60 seconds. Each experiment is repeated 100 times. They are run on two Intel Xeon 5122 CPUs at 3.6 GHz.

C. Results

Table I shows the success rates of the proposed approach and of the constraint planner for the three benchmarks. As the ten trained agents per benchmark have different success rates the maximal, median, average, and minimal success rates are shown. For each benchmark, at least five trained agents achieved a 100% success rate. For the DUAL and PR2 tasks every trained agent is significantly (p-value $< 1\%$) more robust than the constraint planner. No significant difference can be observed on the DELTA task as both approaches have success rates close to 100%. We made the following observations on the failures of both approaches:

Constraint planner: The typical failure mode for the DUAL task of the constraint planner is to get stuck in an axis limit. For the PR2 task the constraint planner is prone to get the PR2's arms tangled and cannot proceed due to collision constraints. It should be noted that the random initial states used for execution are sampled globally within the valid set of states that are within the start mode (not just local disturbances). In the DELTA task the last joint of the delta-robot is continuous and thus has no relevant axis-limit and the collision geometry is mostly convex. For this reason the constraint planner succeeds with high probability during disturbed execution.

Proposed approach: The failures of the proposed approach we observed are due to cyclic switching between different low-level controllers. This occurred mostly during the hand-over of the dual-arm robot, where the trained agent alternates between different controllers that attempt to switch to different grasp modes. These infinite cycles mean that the high-level controller does not "pull through" with a mode switch. This indicates that more sophisticated training methods could further increase robustness.

Table II shows the execution times for the three benchmarks. Additionally the planning times and combined planning and execution times of the constraint planner are shown. The constraint planner is not an optimizing planner and returns the first solution it finds. In contrast the reinforcement learning that forms the high-level controller of our approach aims at a time-optimal policy (for the given set of controllers). If the training had converged to the global optimum one would expect a clear advantage of the proposed approach.

TABLE II

AVERAGE EXECUTION TIME IN SECONDS

Benchmark		DUAL	DELTA	PR2
proposed approach	max	20.1	19.5	19.4
10 different DQNs	median	16.7	14.7	19.2
	mean	16.8	15.1	19.1
	min	13.4	13.0	18.9
constraint	execution time	13.7	16.0	20.8
planner [3]	planning time	3.83	4.58	2.44
	planning + execution	17.6	20.6	23.2

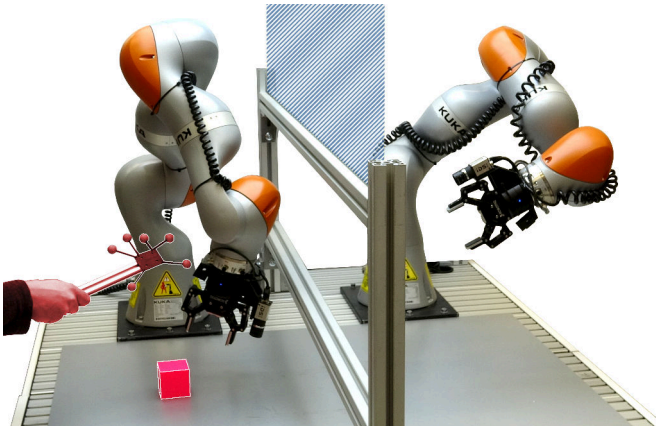

Fig. 6. Real-world experiment with online collision avoidance for the DUAL task: During execution, we add additional constraints to avoid an obstacle that is detected via markers.

The results in Table II show, that this is not consistently the case. For the DUAL and DELTA tasks there are trained agents of the proposed approach with both better and worse average execution times compared to the constraint planner. Only for the PR2 task our approach has consistently better performance. The observed behavior that causes these inefficiencies is again brief oscillations between low-level controllers.

When comparing the execution times of the proposed approach with the combined planning and execution times of the constraint planner the results are, on average, in favor of the proposed approach. It should however be noted that our approach requires that data generation and network training can be done offline ahead of the actual execution. For the benchmarks the combination of data generation and network training took between 4.5 hours and 25 hours.

In addition we conducted two qualitative experiments. As in [3] we can use the low-level controllers to add additional constraints at runtime. We use this for online collision avoidance in the DUAL task on a real robot as shown in Fig. 6. While the high-level controller does not know about these modifications, the low-level controllers will still avoid collisions while respecting the constraints of the task.

As second qualitative experiments we execute the DUAL task in simulation with an additional disturbance. Two seconds after the cube is picked up by the left robot we place it back on the surface to simulate an object that is accidentally dropped. As the high-level controller provides a global mapping between state and active controller it should

react and pick up the dropped object again.

The accompanying video shows all benchmarks as well as the two qualitative experiments. In the qualitative experiments the collision avoidance and the reaction to the dropped part operated as intended.

VI. CONCLUSION

This paper introduced a novel feedback planner for constrained manipulation tasks. Our planner operates on a recently proposed, constraint-based model. From this model a set of constraint-based controllers is derived. This set serves as discrete action-space for a reinforcement learning agent that switches between controllers. The agent is trained offline in simulation and encodes a global mapping of states to robot accelerations for online control.

The advantage of this approach is that the system can react online and in deterministic time to disturbances. This includes reactions to a change of the contact-state, e. g., re-grasping an object that is accidentally dropped. As the constraint-based controllers are derived from the underlying domain model, safety-invariants, such as collision avoidance, are fulfilled by the controllers. This also holds for states to which the reinforcement learning does not generalize. Simulated experiments of three distinct robots and tasks show that the proposed method leads to a significant increase in robustness compared to an existing approach.

A current limitation of our approach is that both action-space and state-space of the reinforcement learning agent grow linearly with the number of discrete modes, which again grows exponentially with the number of manipulated objects. Hence, a promising avenue for future research is to investigate different representations of these spaces as well as policy gradient methods to directly address a continuous action-space.

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] P. S. Schmitt, F. Wirnshofer, K. M. Wurm, G. v. Wichert, and W. Burgard, "Modeling and planning manipulation in dynamic environments," in *Int. Conf. on Robotics and Automation (accepted), preprint available at: http://ais.informatik.uni-freiburg.de/publications/papers/schmitt19icra.pdf*. IEEE, 2019.

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

[5] 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.

[6] W. Decré, H. Bruyninckx, and J. De Schutter, "Extending the itasc constraint-based robot task specification framework to time-independent trajectories and user-configurable task horizons," in *Int. Conf. on Robotics and Automation*. IEEE, 2013, pp. 1941–1948.

[7] 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.

[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] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *Int. Conf. on Robotics and Automation*. IEEE, 2009, pp. 625–632.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.

[21] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *arXiv preprint arXiv:1808.00177*, 2018.

[22] 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.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[24] 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.

[25] 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.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Int. Conf. on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[27] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *Int. Conf. on Learning Representations*, 2016.

[28] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Int. Conf. on Machine Learning*, 2016, pp. 1995–2003.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Int. Conf. on Learning Representations*, 2015.