Dual Execution of Optimized Contact Interaction Trajectories

Marc Toussaint[1] Nathan Ratliff[1,2] Jeannette Bohg[2] Ludovic Righetti[2] Peter Englert[1] Stefan Schaal[2]

***Abstract*— Efficient manipulation requires contact to reduce uncertainty. The manipulation literature refers to this as funneling: a methodology for increasing reliability and robustness by leveraging haptic feedback and control of environmental interaction. However, there is a fundamental gap between traditional approaches to trajectory optimization and this concept of robustness by funneling: traditional trajectory optimizers do not discover force feedback strategies. From a POMDP perspective, these behaviors could be regarded as explicit observation actions planned to sufficiently reduce uncertainty thereby enabling a task. While we are sympathetic to the full POMDP view, solving full continuous-space POMDPs in high-dimensions is hard. In this paper, we propose an alternative approach in which trajectory optimization objectives are augmented with new terms that reward uncertainty reduction through contacts, explicitly promoting funneling. This augmentation shifts the responsibility of robustness toward the actual execution of the optimized trajectories. Directly tracing trajectories through configuration space would lose all robustness—*dual execution* achieves robustness by devising force controllers to reproduce the temporal interaction profile encoded in the *dual* solution of the optimization problem. This work introduces dual execution in depth and analyze its performance through robustness experiments in both simulation and on a real-world robotic platform.**

I. Introduction

Trajectory optimization has become a standard and efficient method for robot motion generation. However, for robot manipulation that requires constraint interaction with objects and obstacles (e.g. sliding before grasping) there still is a crucial gap between trajectory optimization frameworks and the execution of the manipulation in real world. Typically, the optimized trajectory in configuration (joint angle) space is viewed as a reference which should be followed using a controller that corrects for perturbations in configuration space. This approach is successful when deviations between the environmental model used during optimization and the real world are small, but they display essentially no robustness to more substantial variations.

A first approach to circumventing this problem is to try to *model the environment more precisely.* Recently there has been great progress in modeling and simulating contact interactions much more accurately, e.g. using Linear Complementarity Problem (LCP) solvers or related contact models [1], [2], [3], leading to incredibly impressive trajectories for sequential manipulation [4]. However, we believe it is absolutely non-trivial to actually execute them in real-world environments.

[1] `<first>.<last>@informatik.uni-stuttgart.de`
[2] `<first>.<last>@tuebingen.mpg.de`
This work was supported by the EU-ICT Project 3rdHand 610878.

A second approach is to *directly optimize reactive controllers in a full POMDP (dual control) setting.* In this case, we explicitly represent all uncertainties that arise when mapping from a model to the real world and optimize for reactive policies that act optimally under such uncertainty, including the information seeking behavior that is implicit in optimal POMDP policies (e.g. moving to feel the position of an uncertainly placed object). However, this approach can be considered as even more challenging than the previous one. We additionally require exact modeling of real-world uncertainties and face great challenges in POMDP planning. The next section discusses such approaches in detail.

A third approach is to *acknowledge that what we optimize can never truly match the real world.* The optimization problem we formulate—including the model of the environment, the robot's kinematics and dynamics, and the behavior we will see at the control level—can only be an approximation or abstraction of what we see in reality. Therefore, the optimization result cannot serve as a literal reference in the real world. Instead, the onus lies in the *execution*, which becomes a non-trivial interpretation of the optimization result and a translation of its suggested behavior to a reactive feedback controller for the real world. This view of execution introduces a much broader tool set to our choice of abstraction in optimization and corresponding execution.

This paper follows the last approach to develop techniques for leveraging trajectory optimization to solve manipulation tasks that involve direct force interaction with constraints. More specifically:

- We propose specific cost terms in a constrained trajectory optimization setup that mimic an information seeking behavior: they implicitly reward establishing contact with objects that have uncertain pose. This term effectively promotes funneling behaviors within trajectories by steering into constraints, anticipating that the real world system's controllers will use force feedback to re-calibrate the task spaces (object poses) on the fly during execution.
- We employ an extension of the Augmented Lagrangian method to cope with the inequality-constrained trajectory optimization problem. The inner loop solves the unconstrained problem with an efficient Gauss-Newton method exploiting the band-diagonal structure in the Hessian, leading to optimization times below a second in our scenarios.
- We propose executing the optimization result using a feedback controller on the dual variables of the solution, a technique we call *dual execution*. Rather than focus on the configuration space trajectory (the problem's

primal solution), our approach emphasizes reproducing the optimized profile of constraint interactions implicitly encoded within the dual solution of the problem.

In the next section, we review in detail some of the related work that motivates our approach. Section IV-A reviews our general trajectory optimization setup and we introduce an efficient state augmentation for the optimization in Section IV-B that mimics an entropy reduction process in the belief over environmental object poses during constraint interaction. This augmentation enables us to propose a cost term that rewards establishing contact with objects. Section V then discusses the dual execution of the optimization result, where we describe the objective function of a low-level operational space controller that combines desired variables in the dual space, the end-effector's force space, and Cartesian position space. Dual execution is a meta-controller that combines the online (force) feedback and the optimization results to define the desired variables for this low-level controller. Section VI demonstrates this approach both in simulation and on a real-world robot platform.

II. Related work

A. Exploiting contacts for manipulation

Our approach is motivated by a long history of results emphasizing the benefit of exploiting contacts and environmental constraints during manipulation. In [5] the importance of first establishing and then maintaining contact (sliding) is made explicit for problems such as peg-in-hole. The DARPA ARM challenge [6] has singled out the importance of contact: several teams have published reports emphasizing that deliberate contact with the environment was key to the robust manipulation of a diverse range of objects [7], [8], [9]. Similarly, Deimel et al. [10] presented an insightful study on how humans exploit tactile feedback during grasping, especially when their vision is impaired.

All these works motivate the importance of exploiting contact with the environment to generate "funnels", which we understand here as overall system dynamics that reduce uncertainty. The above mentioned approaches usually do not use trajectory optimization methods to design these contact strategies but instead design feedback controllers based on expert knowledge. This work aims to integrate the idea of exploiting contacts with efficient trajectory optimization.

B. Belief Planning

Planning under full POMDP formulations leads to policies that reduce task-relevant uncertainty [11]. While solving POMDPs is hard in general, several approaches have recently been very successful in realizing belief planning in robotics domains: Hsiao et al. [12] derive information seeking "tactile exploration" policies from a POMDP framework, and translated these ideas to robust grasp strategies [13]. Interestingly, Hsiao et al. [14] also explicitly consider the *execution* problem—but given a complete optimized belief plan. More recently, efficient approaches for belief planning in (locally approximated) Gaussian belief space using iLQG have been proposed [15], [16]. However, the running costs is still $O(n^6)$ in the configuration space dimension. Typically, these approaches are used for obstacle *avoidance* [17] instead of seeking contact to reduce uncertainty.

Besides computational complexity, a full POMDP formulation is also challenging as it requires exact modeling of real-world uncertainties. Our approach aims for a simpler integration of information seeking contact behavior with trajectory optimization, by proposing objective function terms in standard (constrained) optimization that mimic the information seeking effect for belief planning.

C. (Constrained) Trajectory optimization

It is beyond the scope of this discussion to review the field of trajectory optimization. Recent methods that address optimizing trajectories involving contact during walking and manipulation are, e.g. [1], [18], [3]. These methods typically involve solving Linear Complementarity Problem (LCP) for closed kinematic chains and leveraging sequential quadratic programming reductions. We will equally employ classical optimization methods, proposing a slight extension of Augmented Lagrangian [19] to efficiently reduce an inequality-constrained optimization problem to a series of unconstrained problems.

D. Robust execution of pre-planned paths

Designing controllers to execute a pre-planned path is a classical topic in robotics. While the naïve approach directly aims to reproduce the planned configuration space trajectory, several classical methods relax this by allowing for an adaptive timing [20], [21], [22]. A further step to make execution more robust is to only reproduce the plan in a select task space, which facilitates exploiting online feedback to influence the null space (e.g. for moving obstacle avoidance) or re-calibrating the take space on the fly [23]. Dual execution integrates these ideas, but focusing on the dual solution instead of a task space reproduction.

III. Dual Execution: Overview

We start by defining the notion of *Dual Execution* as:

> Executing the *dual* solution to a trajectory optimization problem in the real world by designing feedback controllers that track the constraint interaction profile of the optimized trajectory.

The word "dual" here does not refer to other notions like dual control or Kalman duality [24], [25], but to the dual solution.

Let $x(t) \in \mathbb{R}^n$ be the optimized joint configuration of an n-DoF robot. Typically certain task spaces are relevant to a given task, which are defined by a task map $\phi : \mathbb{R}^n \to \mathbb{R}^d$, where $\phi(x)$ may represent end-effector coordinates, distances to objects, relative orientations, or a concatenation of such kinematic features. Task-space execution aims to reproduce the task space trajectory $y(t) = \phi(x(t))$ during the real-world execution.

Now assume that the trajectory is the result of a constrained optimization problem, where in each time slice we have d_g constraints represented by a constraint function

$g:\ \mathbb{R}^n \to \mathbb{R}^{d_g}$ (more details below). The dual solution will then be a d_g-dimensional trajectory $\lambda(t) \in \mathbb{R}^{d_g}$ of Lagrange multipliers. This dual trajectory captures the temporal profile of "interaction" with the constraints, that is, exactly when each constraint is active along the optimal solution. In our case, the constraint functions g will describe the contact of an end-effector with an object in the environment, e.g. a table. In attempting to reproduce the dual trajectory $\lambda(t)$ during real-world execution, we can reproduce the temporal profile of contact with these objects.

This core idea of dual execution means that we can formulate optimization problems that explicitly involve contact with the environment—that, in fact, reward contact with the environment due to the implied information gain—and have controllers that aim to reproduce this optimized contact profile rather than the joint configuration trajectory.

In the following, we will first give technical background on the constrained optimization framework we propose and discuss how to formulate objective functions that will lead to trajectories that seek for contacts. Thereafter, we focus on the execution control itself.

IV. Optimization and Objective Functions that Reward Funneling

A. Optimization Problem Formulation

In this section, we briefly cover our optimization framework—more details on this and our application of the Augmented Lagrangian method can be found in supplementary material.[1]

Let $x_t \in \mathbb{R}^n$ be a joint configuration and $x = x_{1:T} = (x_1, \dots, x_T)$ a trajectory of length T. We consider the optimization problem

$$\begin{aligned} \min_x \quad & \textstyle\sum_{t=0}^T f_t(x_{t-k:t})^\top f_t(x_{t-k:t}) \\ \text{s.t.} \quad & \forall_t:\ g_t(x_t) \le 0\ , \end{aligned} \tag{1}$$

where $x_{t-k:t} = (x_{t-k}, .., x_{t-1}, x_t)$ are $k+1$ tuples of consecutive states, $f_t(x_{t-k:t}) \in \mathbb{R}^{d_t}$ are arbitrary first-order differentiable non-linear k-order cost terms, and $g_t(x_t) \in \mathbb{R}^{m_t}$ are m_t non-linear inequality constraints for each t (totaling $M = \sum_t m_t$ constraints in all). By appropriately defining the k-order cost terms f_t, we can optimize for squared torques and arbitrary tasks in each time slice—see the supplementary material for details[2].

For both f_t and g_t, we assume to have access to their Jacobians, but not the Hessians.

Instead of using log-barrier methods we choose to modify standard Augmented Lagrangian (AL) method [26] to account for inequalities.[2] Generally, AL leads to better conditioned unconstrained problems than log-barriers. The details of this are given in the supplementary material.[1]

[1]http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/optim.pdf

[2] While [26] reduce inequality constraints to simpler, but higher-dimensional, bound-constrained problems (that in turn still require specialized bound-constrained solvers), our approach retains the original problem dimensionality and can leverage in the inner loop traditional unconstrained solvers.

Each inner loop optimization of the Augmented Lagrangian method solves an unconstrained nonlinear optimization problem. The specific form of our problem given by Equation 1 results in a generalized least-squares inner loop optimization that can be solved using a Gauss-Newton method, and specifically one with a *banded* symmetric positive-semidefinite pseudo-Hessian $R = 2\nabla\psi^\top\nabla\psi$ with band-width $(k{+}1)n$ across the full trajectory $x_{0:T}$, where $n = \dim(x_t)$. The banded-ness is crucial for efficient computation of the Newton step and has its origin in the k-order chain structure of Equation 1. The inversion of R (or rather, computation of $\Delta = -R^{-1}\nabla\psi^\top\psi$) is very efficient using appropriate matrix packings and well-studied band-diagonal solvers. In fact, the computational complexity of such a Gauss-Newton step is identical to that of a Kalman or Riccati sweep. In practice, we damp the Gauss-Newton iterations following the Levenberg-Marquardt methodology [26] if non-linearities in f_t or g_t lead to non-decreasing steps.

B. Objective functions that reward (uncertainty) funneling and contact interaction

The inequality constraints g_t correspond to contacts with objects in the environment. The dual solution represents when constraints are active, that is when the solution is in contact. While typical optimization approaches want to *avoid* contact, here we want to propose cost terms that explicitly *favor* funneling behavior by touching or sliding along constraints.

For this we propose two cost terms in this section. First, we define a simple potential that pulls the trajectory into constraints, thereby rewarding sliding contact in general. Second, we derive a method that augments the state space x by a single scalar uncertainty variable (for each constraint), defining a form of uncertainty dynamics that describe how uncertainty decreases with constraint contact, and then planning trajectories that reach a final state with low uncertainty.

1) Potentials pulling into constraints: The first approach is very simple: to the cost at each time slice $f_t(x_{t-k:t})$ we append a term

$$-g(x_t) + \alpha \tag{2}$$

which implies a squared potential $(g(x_t) - \alpha)^2$ for each t in the overall objective, which pulls the value of g towards the positive (constraint violating) parameter α. In the Karush-Kuhn-Tucker (KKT) conditions, this term induces (via ∇_f) exactly the same terms as the Lagrange terms—but pulling into the constrained with "force" $2\alpha^2(g(x) - 1)$ instead of pushing out with force λ. We will demonstrate this objective later.

2) Abstracted uncertainty dynamics and planning towards low final uncertainty: The second approach augments the state x_t with a single scalar (for each constraint) $b \in \mathbb{R}$ which represents a subjective degree of uncertainty. Let us first consider a generic form of uncertainty dynamics before discussing its semantics: We consider

$$\dot b = -\xi(x, b)\ b + b_0 \tag{3}$$

which states that uncertainty reduces with a rate $\xi(x, b)$ that depends on the current configuration (in real-world: sensor feedback) but increases continuously with a small drift b_0. This can be seen in analogy to the evolution of the variance in a Kalman filtering process: b_0 captures a permanent increase in variance as in the prediction step; $-\xi(x, b)\ b$ captures a variance reduction as in the likelihood ("Kalman gain") step. In fact, choosing $\xi(x, b) = (a/(b+a)-1.)$ exactly reproduces the variance reduction of a 1D Gaussian if multiplied with a Gaussian observation of variance a (for instance $a \to \infty$ implies no reduction in variance b). In the following, we consider $b \ll a$ and therefore drop the dependence of $\xi(a, b)$ on b.

Following expert knowledge, we propose to assume that uncertainty reduces whenever the robot is near a constraint.[3] We translate this into the following simple type of uncertainty reduction rate,

$$\xi(x_t) = \alpha\ \sigma(g(x_t)/m + 1)\ , \quad \partial_x \xi = \alpha\ \sigma(1-\sigma)/m\ \partial_x g \tag{4}$$

where m is a margin parameter, α a decay parameter, and σ is the sigmoid function. For $g = 0$, b decays fast with $\approx \alpha$; for $g = -m$, b decays slower with $\alpha/2$; for $g = -2m$, b decays slowly. In brief: only if we are near, or ideally at the constraint, uncertainty reduces fast.

Eqs. (3) and (4) together define the uncertainty dynamics. We augment the state x_t with the uncertainty scalar and enforce the uncertainty dynamics in the trajectory optimization. We do so by penalizating deviations from these dynamics, by appending to time slice costs $f_t(x_{t-k:t}, b_{t-k:t})$ a term proportional to

$$[b_t - (1 - \tau\xi(x_{t\text{-}1}))\ b_{t\text{-}1} - \tau b_0]\ . \tag{5}$$

The Jacobians $\partial_{x_{t\text{-}1}, b_{t\text{-}1}, b_t} f_t$ are straight-forward.

Finally, now that we have the uncertainty simply as part of the state, we can plan trajectories to reach any desired uncertainty state, in particular low uncertainty. We do so by appending a term b_T/σ_b to the final time cost term f_T, which implies a simple additional squared penalty for $b_T > 0$. This second approach is an instantiation of the idea of funneling—here in the sense of using contacts to explicitly funnel uncertainty.

The effects of using the squared potentials vs. planning with the uncertainty dynamics is demonstrated and discussed for the most basic 1D case in Section VI-A.

V. Dual Execution & the Control Architecture

In the previous section, we described how we can define optimization objectives that favor contact and information gain behavior and yield a primal $x(t)$ and dual $\lambda(t)$ solution. We now address the question of how to execute these plans in a way that is (online) adaptive to environmental changes not accounted for during optimization—the core idea is to aim to reproduce the constraint contact pattern implied by the dual solution $\lambda(t)$.

[3]This could be made more formal: considering the information gain of a truncation of a Gaussian that arises from binary tactile feedback [27]; but this is beyond the scope of the current presentation.

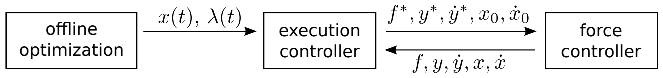


Fig. 1. Assumed control architecture: The offline optimizer returns primal and dual trajectories $x(t), \lambda(t)$; the execution control loop ($\sim$100Hz) translates these, together with information on the currently sensed force f, end-effector $y, \dot{y}$ and joint configuration state $x, \dot{x}$, into desired force, end-effector and nullspace references $f^*, y^*, \dot{y}^*, x_0, \dot{x}_0$ that the operational space/force controller ($\sim$1kHz) uses to compute motor commands.

We generally assume a hierarchical control architecture distinguishing between the outer loop *execution controller* running with about 20-100Hz on a non-real time machine, and an inner loop operational space *position and force controller* running at 1kHz on a real-time machine connected to the robot, see Figure 1. We believe this two-level control structure is appropriate in most realistic robotic systems: Every robotic system has its own low-level motor control infrastructure, being tuned to robustness by engineering expertise—this is certainly true for our real-world robotic system, see below. Our approach does not aim to change or replace this low-level control infrastructure but work with any interface as long as it allows us to send desired positions and interaction forces.

In the following we describe two different low-level force controllers: the first being the actual control infrastructure running on the Apollo robot that we use for our experiments, the second an idealized operational space controller that illustrates more formally our idea about how force control is realized. These descriptions clarify the interface to the execution controller, which we describe last.

A. *Force controller on the Apollo robot*

On the real robot, we use the existing position and force control architecture as described in [9] that allows to control both positions and forces in operational space with good tracking performance. It consists of an inverse dynamics controller with joint position control augmented by a position and force controller in operational space (using Jacobian transpose control laws).

The desired joint position, velocities and accelerations are computed through inverse kinematics given the desired position and velocity reference $y^*, \dot{y}^*$ in task space and an optional nullspace motion $\dot{x}^*$. They are then sent to the inverse dynamics and joint PD controllers. In addition, the task space feedback controller ensures accurate tracking of task space positions and forces.

The controller allows to set a desired end-effector force vector $f^* \in \mathbb{R}^3$ along arbitrary directions. The feedback controller then ensures the tracking of this reference force in operational space while positions are tracked in the other directions. The details of the control architecture and a discussion on its performance can be found in [9].

In conclusion, the controller allows us to set online with high frequency a desired end-effector force f^*, desired end-effector motion $y^*, \dot{y}^*$, and optional nullspace motion $\dot{x}^*$, in this hierarchical order. The execution controller sends these control objectives at a lower frequency. Therefore, we use splines to interpolate between them and ensure smooth control commands.

B. Idealized operational space force controller

For our simulation experiments we choose to have a simpler controller, allowing reproducibility. A general form of operational space force control for fully actuated systems can be described via the objective function

$$F(\ddot{x}) = \|M\ddot{x} + h + J_g^\top \lambda^*\|_H^2 + \|J_\phi \ddot{x} - c\|_C^2 + \|J_g \ddot{x} - b\|_B^2 \tag{6}$$

$$\ddot{x}^* = (M^\top H M + J_g^\top B J_g + J_\phi^\top C J_\phi)^{-1} \times \\ [-M^\top H(h + J_g^\top \lambda^*) + J_g^\top B b + J_\phi^\top C c] \ .$$

The first term is a squared penalty of motor torques $u = M\ddot{x} + h + J_g^\top \lambda^*$ where M is the inertia matrix, h gravity and Coriolis forces, and λ^* desired constraint contact forces translated into joint space via the constraint Jacobian $J_g^\top$. The second term, by choosing $c = \ddot{y}^* - \dot{J}_\phi \dot{x}$, is a squared penalty of desired accelerations in a task space defined by ϕ. The third term, by choosing $b = \ddot{g}^* - \dot{J}_g \dot{x}$, is a squared penalty of desired accelerations along the constraint gradient. The optimum $\ddot{x}$ can be rewritten using Woodbury identities to yield the more common equations of operational space control as special case and allow for precision limits $B \to \infty$ (optionally also $C \to \infty$), so that we can analytically impose the contact constraint. Also a desired nullspace acceleration $\ddot{x}^*$ can be incorporated—we neglect these details here for brevity.

In addition to Equation (6), our simulated controller uses PD feedback to determine the desired accelerations $\ddot{y}^*$ from the current state and the targets $(y^*, \dot{y}^*)$ commanded by the execution controller (equivalently for $\ddot{g}^*$ and $\ddot{x}^*$). The simulated controller assumes touch feedback: when sensing contact, the commanded end-effector force f^* is directly translated to λ^* and $\ddot{g}^*$ is zeroed. In simulation, this will induce the desired force as we assume a correct M and h. Out of contact, λ^* is zeroed and the execution controller may set references $(g^*, \dot{g}^*)$ to steer the end-effector towards the constraint, see below.

In conclusion, such a type of operational space force controller allows us to define (and online adapt) arbitrary task and constraint spaces via ϕ and g, set desired constraint forces λ^*, and set references in the task spaces $(y^*, \dot{y}^*)$ and $(g^*, \dot{g}^*)$. Our simulation experiments use this controller.

C. Execution controller

Finally, the execution controller has the task of defining the inputs $(f^*, y^*, \dot{y}^*, \dot{x}^*)$ to the operational space force controller at high frequency. The optimizer returned $x(t), \lambda(t)$. Using the known end-effector kinematics ϕ, we compute from this also the end-effector reference $y(t)$. Throughout execution, $x(t)$ will serve as a weak nullspace reference for the force controller.

The aim of execution is to follow the dual reference $\lambda(t)$, that is, to create or maintain constraint contact if $\lambda(t) > 0$ and avoid contact for $\lambda(t) = 0$. We do this by distinguishing four cases—the first one being the most interesting as it also re-calibrates our constraint defining map g based on missing tactile feedback:

- Case $f = 0 \wedge \lambda(t) > 0$: we sense no force while the dual reference commands to be in contact. In this case, we design a PD behavior along the constraint gradient J_g that pulls the end-effector into the constraint. We also send the planned task space reference $y(t)$ which is, due to the hierarchical low level operational space controller, only followed along directions orthogonal to J_g (planar to the constraint).
  Importantly, we use the missing force feedback to online re-calibrate our constraint defining map g: we shift the offset of g to be consistent with the observation of no contact. This will influence the future controller as tasks are defined relative to the constraint.
- Case $f > 0 \wedge \lambda(t) = 0$: we sense force while the dual reference commands to be out of contact. In this case, we impose a PD behavior along the constraint gradient J_g that pushes the end-effector out of the constraint. As above, $y(t)$ operations only orthogonal to J_g.
- Case $f > 0 \wedge \lambda(t) > 0$: we sense force and the dual reference also commands to be in contact. We continue to send a desired contact force f^* (colinear with the constraint Jacobian J_g) to the force controller. As above, $y(t)$ operations only orthogonal to J_g. This will lead to a stable sliding along the constraint.
- Case $f = 0 \wedge \lambda(t) = 0$: we sense no force and the dual reference commands to be out of contact. Only the planned task space reference $y(t)$ is send to the operational space controller.

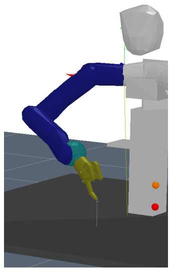
This simple execution control will try to maintain robust contact with a constraint and uses the tactile feedback to calibrate the constraint function g. This approach aims to be robust w.r.t. model errors in g: even if the real-world constraint is differently located to the model used for optimization, the above execution control will robustly reproduce the temporal contact profile implied by the dual reference $\lambda(t)$.

VI. Experiments

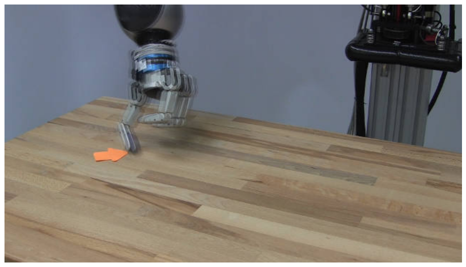
A. Illustrating the effect of objective functions

We briefly illustrate the effect of the contact rewarding objective functions we defined in Section IV-B on the optimized solutions by considering the simplest example. Consider a 1D state $x(t)$ which starts at $x(0) = 1$ and constraints $\forall_t : \ g_t(x) = -x + a$ which enforces $\forall_t : \ x(t) > a$. We set a task goal stating that the final state should be $x(T) \approx a + 0.1$, that is 0.1 (meters) above the constraint. We penalize squared accelerations of the system throughout and the squared deviation $(x(T) - (a + 0.1))^2$ from the task target. For planning, it is assumed that $a = 0$, but during execution in the real-world this might not be the case. We first consider the results of optimization only.

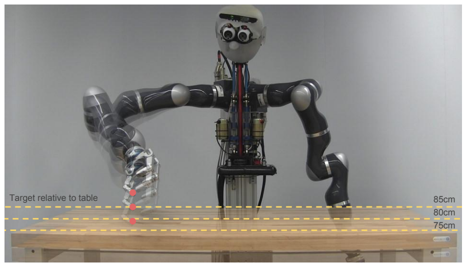
Figure 3(a) shows the optimal trajectory for using no funneling objective. The solution accelerates directly from $x(0) = 1$ to $x(T) = 0.1$. The constraint is never touched;



(a) Simulation

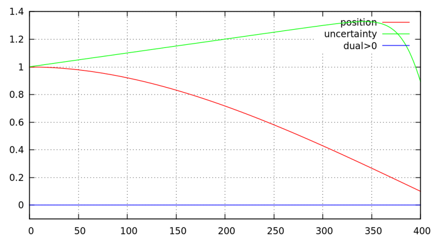


(b) Repeatability of reaching

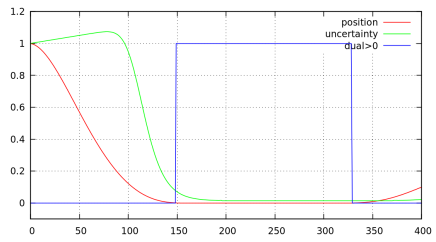


(c) Adaptivity to unknown constraint heights

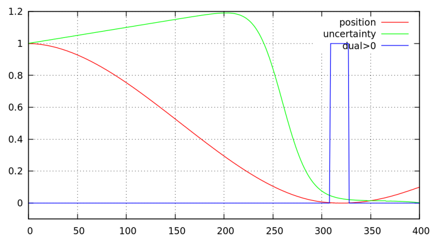
Fig. 2. (a) The setup used for the simulated experiments. (b) The real Apollo robot, with 7 DoF Kuka arm, reaching three times to a target on the table at constant height. (c) Apollo reaching to a target that is defined relative to the table (red dot). Despite varying, initially unknown table heights as indicted by the dashed lines, the reaching behavior is robustly adapted online through force feedback.



(a) No reward of information gain



(b) Potential pulling into the constraint



(c) Setting a belief goal

Fig. 3. Position x, uncertainty scalar b and dual solution λ over time for different objective functions to reward constraint contact.

$\lambda_t = 0$ throughout the trajectory; uncertainty mostly increases with a rate b_0. Figure 3(b) shows the optimal trajectory for a squared potential continuously pulling towards the constraint throughout the trajectory. The solution first moves directly to the constraint, slides along the constraint, at the end moves up to $x(T) = 0.1$. The uncertainty exponentially decreases during the contact. Figure 3(c) shows the optimal trajectory when uncertainty funneling is used: a squared potential on the final uncertainty b_T and enforcement of the uncertainty dynamics. The solution touches the constraint later, slides along to sufficiently reduce uncertainty, at the end moves up to $x(T) = 0.1$.

B. Dual execution in simulation

We consider a simulated setup of the Apollo robot with a 7 DoF Kuka arm as illustrated and described in Figure 2(a). In all trials, the true height of the table is unknown to the system, the true target (red ball) is always located 10cm above the table; the yellow ball indicates the current estimate of the target (and implicitly of the table height). To solve this problem reliably, the robot needs to establish contact to locate the table height. We used our optimization framework to generate trajectories $x(t)$ and $\lambda(t)$ which involve a sliding table contact during the middle of the trajectory. The dual reference $\lambda(t)$ is the red curve in Figure 4(a). The computation time for optimization over 200 time slices on a normal laptop with non-optimized code is 0.748 seconds.

We performed 10 runs with and without dual execution, where in the latter case the execution controller simply commands the planned task space trajectory $y(t)$ to the operational space controller. The table height is randomly drawn from a Gaussian with std 0.1 and unknown to the robot. As expected, without dual execution the generated trajectory is always the same, not seeking for tactile feedback and not responding to varying constraint heights. With dual execution enabled, the robot reliably generates contact with the constraint and thereby re-calibrates the task space and reaches the target.

method/scenario	mean error
no dual execution	0.1152
dual exec. w/o oscillations	0.0030974
dual exec. + oscillations	0.021770

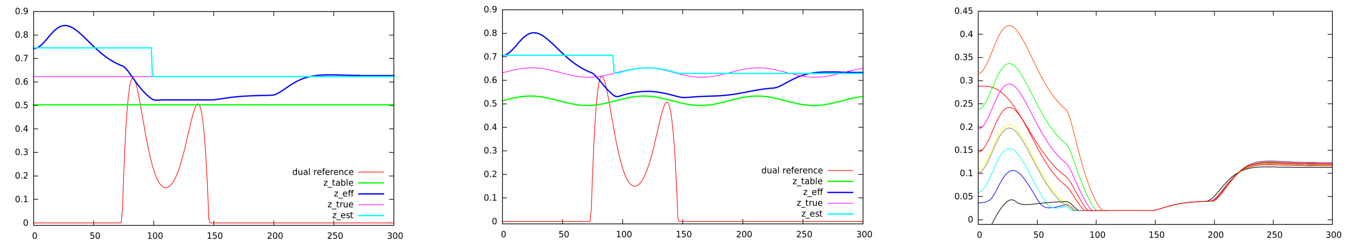
Fig. 5. Square roots of mean squared errors over 10 runs for reaching a target exactly 10cm above the table. The table height is initially unknown/uncertain. (a) not using dual execution (error equals standard deviation of randomized table placements), (b) using dual execution, (c) using dual execution while the table height is oscillating online (example runs given in Figures 4 and the video.

Figure 4 displays exemplary runs and the relative hand-table height for 10 runs. For instance in 4(a) around time 70, we nicely see that the robot was expecting contact ($\lambda(t) > 0$), but did not sense it, and accelerates the end-effector towards the constraint (kink in blue curve). The gap between blue and green during contact is exactly the table thickness. When $\lambda(t) = 0$, the robot releases contact again and eventually reaches the target. Figure 4(b) displays a run where the table was moving (oscillating) online: during contact we nicely see the compliant sliding on the constraint and respective online re-calibration of the estimated target height (oscillating as is the true target).

Figure 5 displays the target errors for all three cases. As expected the error when not using dual execution equals to the standard deviation of target height, independent of oscillating or not.

C. Experiments on a physical platform

We implemented dual execution for a 7-DOF Kuka arm with a 4-DOF Barrett hand and ran a collection of real world



(a) Exemplary run, static constr. (b) Exemplary run, moving constr. (c) hand-table height for 10 runs

Fig. 4. (a) Exemplary run of dual control for a static constrained, displaying the pre-planned dual reference λ, end-effector (z_eff), estimated target (z_est), true table (z_table), and target (z_true) heights over time (in $\frac{1}{100}$sec). (b) Same when the constraint (table height) is oscillating. (c) Relative hand-table height for 10 runs, converging robustly to the target height.

Height	0.75	0.77	0.80	0.82	0.85	0.87	Avg
On Table	0.01	0.009	0.01	0.005	0.014	0.015	0.010
Above Table	0.003	0.01	0.01	0.009	0.012	0.011	0.009

Fig. 7. Average distance of the final cartesian positions to the mean position of three trials per table height and of all trials. (metres)

experiments. We analyzed two scenarios: first, a reaching task with the goal of reaching a particular point *on* a table robustly against unknown variations in the table height; and second, another reaching task with a goal of touching a point hovering above the table, again amidst unknown variations in the table's height.

For each scenario, we collected three trials at six different height settings of the table, amounting to a total of 36 data points. For both problems, the table varied in height across the scenarios by a total of 12 cm.

The dual execution of planned trajectories for these scenarios leads to very robust behaviors that successfully achieve the tasks in both scenarios despite a 12cm deviation in the table height across all trials. Table 7 gives statistical results for the average deviation from the target point achieved across the executions. For each experimental table height, ranging from 75cm to 87cm, Columns 2 through 7 of the table report individual average deviations for the scenarios at those heights, and the final column gives the average deviation computed across all trials collectively for each scenario. Of note, the overall deviation is never more than 1cm. Given that the internal planning model assumed the table to be 80cm high, we would expect a naïve execution of the primal trajectory to not only have much more variation in its final goal error across these table heights, but to also ultimately be unsuccessful in its execution when the table is actually higher than expected.

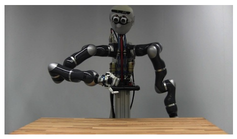
We also, in more detail, profile the qualitative behavior of Scenario 2 (reaching out to a point hovering above the table). Throughout the trials, we can observe five distinct behavioral stages that emerge from the combined plan and dual execution, see Figure 6. Initially, the robot starts off executing the behavior as normal, simply following the trajectory as planned by the optimizer. Around 1.1 seconds into the execution, it realizes that it had planned to have contacted the surface, but it has not yet actually detected contact. At that point, it begins to explicitly steer toward the constraint until at around 3.2 seconds when it finally reaches the surface. From then on out, it is able to successfully exert the desired force for the remainder of the time indicated by the dual solution until at 5 seconds it again lifts off from the constraint to raise to the desired target point hovering above the table.

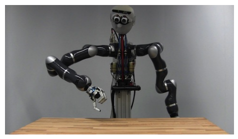
The lower two plots of Figure 6 additionally depict the distribution of data traces for all trials of each scenario, showing both measured contact forces and Cartesian height values relative to the table; each stage of the dual execution behavior can be seen within the data. The red line indicates the point at which the controller believes it should be feeling the force. At that point, we see the height variable dip more aggressively toward the constraint, along with a high variance region of measured forces (between seconds 2 and 3), reflecting differing contact times of the hand across trials of different heights. From there, the distribution over relative hand heights funnels, and we see distinctly more consistent task profiles for the remainder of the plot.

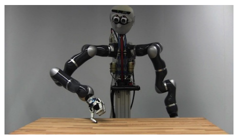
The center and rightmost subplots of Figure 2, visually give a feel for the constancy of the execution across the trials. The center subplot shows the variation in final on-table achieved location for Scenario 1 for three trials at a constant table height, and the rightmost subplot shows the robot consistently ending at the same height relative to the table across trials of *differing* table heights.

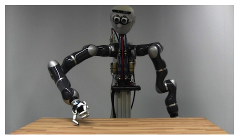
VII. Conclusion

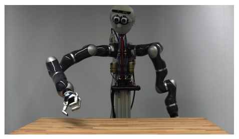
This work was motivated by the understanding that exploiting contacts that generate funnels is critical for effective manipulation. We asked how this insight can be leveraged within efficient trajectory optimization methods. The idea we put forward was to reproduce the dual solution of a constrained trajectory optimization problem during the online execution, which represents the temporal pattern of constraint interaction. To realize this, we first proposed cost terms in the optimization objective that favor trajectories that are in contact, e.g. by conditioning the final uncertainty to be low. Second, we proposed a dual execution controller that tries to reproduce the contact profile during execution of the plan, even when the constraints differ from those used during planning or are moving online. With this we provided novel methods to design contact exploiting manipulation based on trajectory optimization.

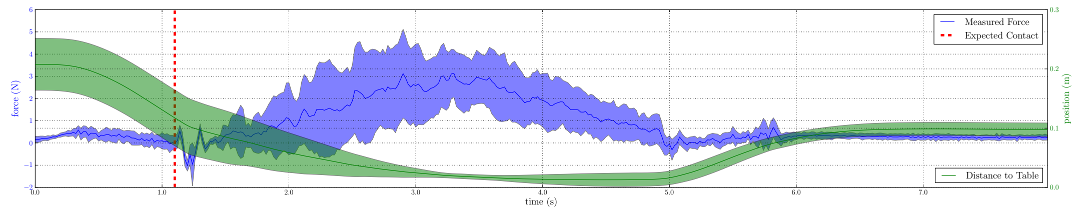

(a) Starting position 0s


(b) Expected contact 1.1s

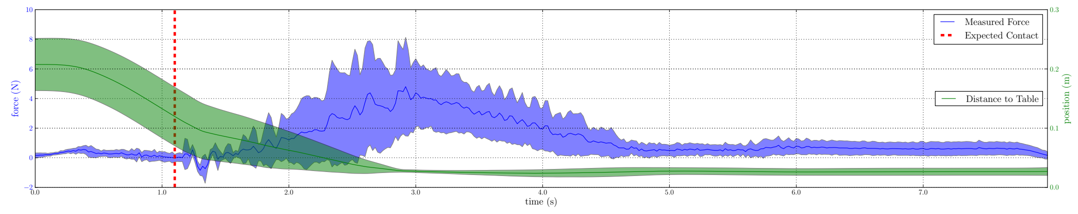

(c) First contact 3.2s


(d) Lifting off 5s


(e) Target reached 6s



(f) Reaching to target *above* the table. For a visualization of the experiment, we refer to Figures 6(a)- 6(e).



(g) Reaching to target *on* the table.

Fig. 6. Measured forces and distance to table during execution of the two reaching experiments. Plots show the mean and standard deviation over 18 runs (three trials per table height variation). Blue: Measured Forces (N). Green: Distance to Table (m).

References

[1] M. Posa and R. Tedrake, “Direct trajectory optimization of rigid body dynamical systems through contact,” in *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 527–542.

[2] E. Todorov, “A convex, smooth and invertible contact model for trajectory optimization,” in *International Conference on Robotics and Automation*. IEEE, 2011.

[3] T. Erez and E. Todorov, “Trajectory optimization for domains with contacts using inverse dynamics,” in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 4914–4919.

[4] I. Mordatch, Z. Popović, and E. Todorov, “Contact-invariant optimization for hand manipulation,” in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA ’12. Eurographics Association, 2012, pp. 137–144.

[5] T. Lozano-Perez, M. T. Mason, and R. H. Taylor, “Automatic synthesis of fine-motion strategies for robots,” *The International Journal of Robotics Research*, vol. 3, no. 1, pp. 3–24, 1984.

[6] “ARM — Autonomous Robotic Manipulation,” http://thearmrobot.com.

[7] N. Hudson, T. Howard, J. Ma, A. Jain, M. Bajracharya, S. Myint, C. Kuo, L. Matthies, P. Backes, P. Hebert, T. Fuchs, and J. Burdick, “End-to-end dexterous manipulation with deliberate interactive estimation,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 2371–2378.

[8] M. Kazemi, J.-S. Valois, J. A. Bagnell, and N. S. Pollard, “Robust object grasping using force compliant motion primitives.” in *Robotics: Science and Systems*, 2012.

[9] L. Righetti, M. Kalakrishnan, P. Pastor, J. Binney, J. Kelly, R. Voorhies, G. Sukhatme, and S. Schaal, “An autonomous manipulation system based on force control and optimization,” *Autonomous Robots*, vol. 36, no. 1-2, pp. 11–30, 2014.

[10] R. Deimel, C. Eppner, J. lvarez Ruiz, M. Maertens, and O. Brock, “Exploitation of environmental constraints in human and robotic grasping,” in *International Symposium on Robotics Research (ISRR)*, 2013.

[11] L. P. Kaelbling and T. Lozano-Pérez, “Integrated task and motion planning in belief space,” *Submitted. Draft at http://people. csail. mit. edu/lpk/papers/HPNBelDraft. pdf*, 2012.

[12] K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez, “Task-driven tactile exploration.” in *Robotics: science and systems*, 2010.

[13] ——, “Robust grasping under object pose uncertainty,” *Autonomous Robots*, vol. 31, no. 2-3, pp. 253–268, 2011.

[14] K. Hsiao, T. Lozano-Pérez, and L. P. Kaelbling, “Robust belief-based execution of manipulation programs,” in *Eighth Intl. Workshop on the Algorithmic Foundations of Robotics*, 2008.

[15] J. Van Den Berg, S. Patil, and R. Alterovitz, “Motion planning under uncertainty using iterative local optimization in belief space,” *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1263–1278, 2012.

[16] R. Platt Jr, R. Tedrake, L. Kaelbling, and T. Lozano-Perez, “Belief space planning assuming maximum likelihood observations,” in *Robotics: Science and Systems (RSS)*, 2010.

[17] A. Lee, Y. Duan, S. Patil, J. Schulman, Z. McCarthy, J. van den Berg, K. Goldberg, and P. Abbeel, “Sigma hulls for gaussian belief space planning for imprecise articulated robots amid obstacles,” in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 5660–5667.

[18] I. Mordatch, Z. Popović, and E. Todorov, “Contact-invariant optimization for hand manipulation,” in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association, 2012, pp. 137–144.

[19] J. Nocedal and S. J. Wright, *Penalty and Augmented Lagrangian Methods*. Springer, 2006.

[20] O. Dahl and L. Nielsen, “Torque-limited path following by online trajectory time scaling,” *Robotics and Automation, IEEE Transactions on*, vol. 6, no. 5, pp. 554–561, 1990.

[21] H. Arai, K. Tanie, and N. Shiroma, “Time-scaling control of an underactuated manipulator,” in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 3. IEEE, 1998, pp. 2619–2626.

[22] J. Z. Kolter, A. Coates, A. Y. Ng, Y. Gu, and C. DuHadway, “Space-indexed dynamic programming: learning to follow trajectories,” in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 488–495.

[23] N. Jetchev and M. Toussaint, “Task space retrieval using inverse feedback control,” in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 449–456.

[24] H. U. Nikolai Michailovich Filatov, *Adaptive Dual Control: Theory and Applications*. Springer, 2004.

[25] R. Kalman, “A new approach to linear ltering and prediction problems,” in *ASME Transactions journal of basic engineering*, 1960.

[26] J. Nocedal and S. J. Wright, *Numerical Optimizatoin*. Springer, 2006.

[27] M. Toussaint, “Pros and cons of truncated gaussian ep in the context of approximate inference control,” in *NIPS Workshop on Probabilistic Approaches for Robotics and Control*, 2009.

Technical Reference: Constrained Trajectory Optimization

Supplementary material for "Dual Execution of Optimized Contact Interaction Trajectories"

Marc Toussaint[1] Nathan Ratliff[1,2] Jeannette Bohg[2] Ludovic Righetti[2] Peter Englert[1] Stefan Schaal[2]

I. Optimization Problem Formulation

In this supplementary material we give more details on the optimization framework. This includes a basic extension of the Agumented Lagrangian method to deal with inequalities. Unlike previously proposed solutions in the literature [1] that reduce inequality constrained problems to simpler, but higher-dimensional, bound-constrained problems (that in turn still require specialized bound-constrained solvers), our approach retains the original problem dimensionality and can leverage in the inner loop traditional unconstrained solvers, such as a straightforward implementation of Newton's method.

Let $x_t \in \mathbb{R}^n$ be a joint configuration and $x = x_{1:T} = (x_1, \dots, x_T)$ a trajectory of length T. We consider the optimization problem

$$\begin{aligned} \min_x \quad & \sum_{t=0}^{T} f_t(x_{t-k:t})^\top f_t(x_{t-k:t}) \\ \text{s.t.} \quad & \forall_t : \; g_t(x_t) \le 0 \; . \end{aligned} \tag{1}$$

where $x_{t-k:t} = (x_{t-k}, .., x_{t-1}, x_t)$ are $k+1$ tuples of consecutive states, $f_t(x_{t-k:t}) \in \mathbb{R}^{d_t}$ are arbitrary first-order differentiable non-linear k-order cost vectors, and $g_t(x_t) \in \mathbb{R}^{m_t}$ are m_t non-linear inequality constraints for each t (totaling $M = \sum_t m_t$ constraints in all).[1]

The k-order cost vectors $f_t(x_{t-k:t}) \in \mathbb{R}^{d_t}$ are very flexible in including various elements that can represent both transition and task-related costs. For instance, for transitional costs, we can penalize square velocities using $k = 1$ (depending on two consecutive configurations) $f_t(x_{t\text{-}1}, x_t) = (x_t - x_{t\text{-}1})$, and square accelerations using $k = 2$ (depending on three consecutive configurations) $f_t(x_{t\text{-}2}, x_{t\text{-}1}, x_t) = (x_t + xt\text{-}2 - 2x_{t\text{-}1})$. Likewise, for larger values of k, we can penalize higher-order finite-differencing approximations of trajectory derivatives. Moreover, for $k = 2$, using the equations of motion $M\ddot{x}_t + F = \tau_t$ with finite-differencing acceleration approximation $\ddot{x}_t \approx x_{t+1} + xt - 1 - 2x_t$, we can explicitly penalize square torques as well using $f_t = \sqrt{H}M(x_t - 2x_{t\text{-}1} + x_{t\text{-}2}) + F)$, where $\sqrt{H}$ is the Cholesky decomposition of a torque cost metric H, implying costs $f_t^\top f_t = u_t^\top H u_t$.

For task costs, using terms of the form $f_t \supset (y^* - \phi(x))/\sigma$ we can induce squared potentials in some task space ϕ. Here, the somewhat awkward notation $f_t \supset v$ means that we constructed the vector f_t by appending v to f_t.

The hard constraints g_t will represent contacts with objects. (It would be trivial to also include equality constraints $h_t(x_t) = 0$ for each t; we neglect this augmentation for simplicity.)

Both, f_t and g_t are non-linear and we assume to have access to their Jacobians, but not the Hessians. This allows us to use Gauss-Newton type methods, where the pseudo Hessian is always semi positive definite.

1) Augmented Lagrangian: While log-barriers have become a standard for interior point methods we consider the alternative Augmented Lagrangian (AL) approach [1] because it naturally copes with infeasible initializations, without a need for a phase I optimization and because AL very directly relates to standard squared penalty methods to cope with task constraints while ensuring exact constraint satisfaction without taking the ∞ penalty limit (or steep barrier limit)—thereby leading to much better conditioned problems.[2] In our experience, the dual solution converges very fast using AL; indeed, if all involved constraints and cost were linear the algorithm converges in just one iteration. In contrast, central path following log-barriers methods does not have this property.

Traditional applications of AL to inequality constraints augment the space with extra slack variables s_t turning inequality constraints of the form $g_t(x) \le 0$ into bound-constrained equality constraints of the form $g_t(x) = s_t$ with $s_t \le 0$. Such reductions increase the problem's dimensionality and require that inner loop problems be solved by more specialized bound-constrained solvers.

We propose an alternative straight-forward extension to AL that retains the original dimensionality of the problem and allows us to leverage generic fully unconstrained solvers in the inner loop. Given the original constrained problem

$$\min_x \quad f(x) \quad \text{s.t.} \quad g(x) \le 0 \; , \quad h(x) = 0 \; . \tag{2}$$

the Augmented Lagrangian method considers the unconstrained problem

$$\begin{aligned} \min_x \quad & f(x) \\ & + \mu \sum_i [g_i(x) \ge 0 \vee \lambda_i > 0] \; g_i(x)^2 + \sum_i \lambda_i g_i(x) \\ & + \mu \sum_i h_i(x)^2 + \sum_{i=1} \nu_i h_i(x) \end{aligned} \tag{3}$$

[1] <first>.<last>@informatik.uni-stuttgart.de
[2] <first>.<last>@tuebingen.mpg.de

[1]The first cost vector $f_0(x_{-k}, .., x_0)$ depends on states x_t with negative t. We call these $(x_{-k}, .., x_{-1})$ the *prefix*. The prefix defines the initial condition of the robot, which we usually assume to be resting at some given x_0.

[2]Lower largest to smallest eigenvalue ratios of the Hessian in the respective unconstrained problems.

with dual parameters λ_i, ν_i and penalty parameter μ. The notation $[\textit{expr.}] \in \{0,1\}$ is the indicator function for a boolean expression. The AL extension thereby combines the standard Lagrange terms with squared penalties in such a way that the squared term $g_i(x)^2$ is in effect only when either the constriant $g_i(x) \le 0$ is violated or the Lagrange multiplier λ_i is positive. In other words, that squared penalty disappears when all the constraints on the problem are satisfied ($\gamma_i(x) \le 0$ and $\lambda_i = 0$), at which point the linear $g_i(x)$ term drops out as well.

At each iteration, we solve this unconstrained problem and then update

$$\nu_i \leftarrow \lambda_i + 2\mu h_i(x') \tag{4}$$

$$\lambda_i \leftarrow \max(\lambda_i + 2\mu g_i(x'), 0) \ . \tag{5}$$

With this choice of dual variables the Lagrange terms will, in the next iteration, generate exactly the gradients that were previously generated by the squared penalties. It is straight-forward to show that if f, g and h were linear, after the first iteration the dual parameters are the optimal dual solution, the primal solution to (3) is feasible and optimal, and the squared penalty terms become zero. Clearly, $\lambda_i \ge 0$ is guaranteed to be non-negative; the expression $[g_i(x) \ge 0 \vee \lambda_i > 0]$ activates the squared penalty only if the inequality is violated or λ_i has been active (non-zero) in the previous iteration.

2) Gauss-Newton: To solve the unconstrained problem (3) we use Gauss-Newton: Note that the specific form (1) of our optimization problem will lead to a *banded* symmetric semi pos-def. pseudo-Hessian $R = 2\nabla\psi^\top\nabla\psi$ with band-width $(k\text{+}1)n$, over the full trajectory $x_{0:T}$. Here $\psi = (f_0^\top, .., f_T^\top)^\top$ is the concatenation of all f_t over all time slices. The banded-ness is crucial and has its origin in the k-order chain structure of the cost vectors f_t in (1). The inversion of R (or rather, computation of $\Delta = -R^{\text{-}1}\nabla\psi^\top\psi$) is very efficient using appropriate matrix packings and well-studied band-diagonal solvers. In fact, the computational complexity of such a Gauss-Newton step is identical to that of a Kalman or Riccati sweep. In practice, we damp the Gauss-Newton iterations following the Levenberg-Marquardt methodlogy if non-linearities in f_t or g_t lead to non-decreasing steps.

REFERENCES

[1] J. Nocedal and S. J. Wright, *Numerical Optimizatoin*. Springer, 2006.