2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

Constrained Differential Dynamic Programming Revisited

Yuichiro Aoyama[1,2], George Boutselis[1], Akash Patel[1] and Evangelos A. Theodorou[1]

Abstract— **Differential Dynamic Programming (DDP) has become a well established method for unconstrained trajectory optimization. Despite its several applications in robotics and controls, however, a widely successful constrained version of the algorithm has yet to be developed. This paper builds upon penalty methods and active-set approaches towards designing a Dynamic Programming-based methodology for constrained optimal control. Regarding the former, our derivation employs a constrained version of Bellman's principle of optimality, by introducing a set of auxiliary slack variables in the backward pass. In parallel, we show how Augmented Lagrangian methods can be naturally incorporated within DDP, by utilizing a particular set of penalty-Lagrangian functions that preserve second-order differentiability. We demonstrate experimentally that our extensions (individually and combinations thereof) enhance significantly the convergence properties of the algorithm, and outperform previous approaches on a large number of simulated scenarios.**

I. Introduction

Originally developed by Jacobson and Mayne, Differential Dynamic Programming (DDP), one of the most successful trajectory optimization algorithms [1], have been employed in various applications such as robotic manipulation [2], bipedal walking [3] and model-based reinforcement learning [4], to name a few. DDP is an indirect method which utilizes Bellman's principle of optimality to split the problem into "smaller" optimization subproblems at each time step. Under mild assumptions on the cost and dynamics, it can be shown that DDP achieves locally quadratic convergence rates [5]. While the original method relies on second-order derivatives, one of its variations, iterative-Linear-Quadratic-Regulator (iLQR), uses only Gauss-Newton approximations of the cost Hessians as well as first-order expansions of the dynamics [6], which is often numerically advantageous.

While unconstrained DDP has been widely tested and used over the past decades, its constrained counterpart has yet to be properly established. Since most practical applications in controls and robotics include state and/or control constraints (e.g., navigating through obstacles, respecting joint/actuator limits, etc.), off-the-shelf optimization solvers still remain the most popular tool for trajectory optimization among scientists and practitioners [7], [8]. The main drawback of using these optimization solvers is that feasibility with respect to dynamics has to be explicitly imposed, thus slowing down the optimization process. A few works have attempted to extend the DDP framework to the constrained case. [9], [10] considered the case of control bounds by solving several quadratic programs over the trajectory, and [11] dealt with equality constraints only via projection techniques. The works in [12], [13] utilized the Karush-Kuhn-Tucker (KKT) conditions when both state and control constraints are present, with [13], [14], in particular, solving successive quadratic programs in the forward pass of the method. However, these methods tend to be susceptible to bad initializations and not robust enough. [12], [15], [16] also discussed combining DDP with an Augmented Lagrangian (AL) approach; [12] updated the Lagrange multipliers via forward/backward passes, while [15], [16] utilized schemes from the standard Powell-Hestenes-Rockafellar (PHR) methodology [17] with first-order approximations of the Hessians. We find that such AL-based methods typically take long iterations to give strictly feasible trajectories, since constraints are treated as a part of the cost.

In this paper we build upon the works in [12], [13], [15] to develop a state- and control-constrained version of DDP in discrete time. Specifically, we extend [12], [13] by introducing a slack variable formulation into Bellman's principle, and thus avoid assumptions regarding the active constraints of the problem. Moreover, we propose an AL-inspired algorithm, by considering a set of penalty functions that preserves smoothness of the transformed objective function. This property was not satisfied in [15], but is required to establish the convergence properties of DDP [5]. These two methodologies can be used separately, or be properly combined for improved numerical performance. To the best of the authors' knowledge, such an extensive experimental study on constrained trajectory optimization has not been conducted in the past. We believe that the current work is a key step towards the development of a numerically robust constrained DDP, opening up multiple directions for research and further improvements.

II. Preliminaries

A. Unconstrained Differential Dynamic Programming

We will briefly cover here the derivation and implementation of DDP. More details can be found in [1], [6].

Consider the discrete-time optimal control problem

$$\min_{\boldsymbol{U}} J(\boldsymbol{X},\boldsymbol{U}) = \min_{\boldsymbol{U}} \left[\textstyle\sum_{k=0}^{N-1} l(\boldsymbol{x}_k,\boldsymbol{u}_k) + \phi(\boldsymbol{x}_N)\right] \quad (1)$$
$$\text{subject to} \quad \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k,\boldsymbol{u}_k),\ k=0,...,N-1.$$

where $\boldsymbol{x}_k \in \mathbb{R}^n$, $\boldsymbol{u}_k \in \mathbb{R}^m$ denote the state and control input of the system at time instant t_k, respectively, and $\boldsymbol{f}: \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ corresponds to the transition dynamics function. The scalar valued functions l, ϕ, J, denote the running, terminal and total cost of the problem, respectively.

[1]School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA, USA
[2]Komatsu Ltd.,Tokyo, Japan
{yaoyama3, gbouts, apatel435, evangelos.theodorou}@gatech.edu

We also let $\boldsymbol{X} := (\boldsymbol{x}_0^\top, \ldots, \boldsymbol{x}_N^\top)$ and $\boldsymbol{U} := (\boldsymbol{u}_0^\top, \ldots, \boldsymbol{u}_{N-1}^\top)$ be the state/control sequences over the horizon N.

Of paramount importance is the concept of the *value function*, representing the minimum cost-to-go at each state and time, defined as:

$$V_k(\boldsymbol{x}_k) := \min_{\boldsymbol{u}_k} J(\boldsymbol{X}, \boldsymbol{U}). \tag{2}$$

Based on this, *Bellman's principle of optimality* gives the following rule:

$$V_k(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k}[l(\boldsymbol{x}_k, \boldsymbol{u}_k) + V_{k+1}(\boldsymbol{x}_{k+1})]. \tag{3}$$

DDP finds local solutions to (1) by expanding both sides of (3) about given nominal trajectories, $\bar{\boldsymbol{X}}$, $\bar{\boldsymbol{U}}$. Specifically, let us define the Q function as

$$Q_k(\boldsymbol{x}_k, \boldsymbol{u}_k) := l(\boldsymbol{x}_k, \boldsymbol{u}_k) + V_{k+1}(\boldsymbol{x}_{k+1}). \tag{4}$$

We now proceed by taking quadratic expansions of Q_k about $\bar{\boldsymbol{X}}$, $\bar{\boldsymbol{U}}$. According to (4) and an additional expansion of $\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k)$, this will give

$$\begin{aligned}Q_k(\boldsymbol{x}_k, \boldsymbol{u}_k) &\approx Q_k + Q_{\boldsymbol{x},k}^\top \delta\boldsymbol{x}_k + Q_{\boldsymbol{u},k}^\top \delta\boldsymbol{u}_k + \\ &\tfrac{1}{2}(\delta\boldsymbol{x}_k^\top Q_{\boldsymbol{xx},k}\delta\boldsymbol{x}_k + 2\delta\boldsymbol{x}_k^\top Q_{\boldsymbol{xu},k}\delta\boldsymbol{u}_k + \delta\boldsymbol{u}_k^\top Q_{\boldsymbol{uu},k}\delta\boldsymbol{u}_k),\end{aligned}$$

with

$$\begin{aligned}&Q_{\boldsymbol{xx},k} = l_{\boldsymbol{xx}} + \boldsymbol{f}_{\boldsymbol{x}}^\top V_{\boldsymbol{xx},k+1}\boldsymbol{f}_{\boldsymbol{x}},\ Q_{\boldsymbol{x},k} = l_{\boldsymbol{x}} + \boldsymbol{f}_{\boldsymbol{x}}^\top V_{\boldsymbol{x},k+1} \\ &Q_{\boldsymbol{uu},k} = l_{\boldsymbol{uu}} + \boldsymbol{f}_{\boldsymbol{u}}^\top V_{\boldsymbol{xx},k+1}\boldsymbol{f}_{\boldsymbol{u}},\ Q_{\boldsymbol{u},k} = l_{\boldsymbol{u}} + \boldsymbol{f}_{\boldsymbol{u}}^\top V_{\boldsymbol{x},k+1} \\ &Q_{\boldsymbol{xu},k} = l_{\boldsymbol{xu}} + \boldsymbol{f}_{\boldsymbol{x}}^\top V_{\boldsymbol{xx},k+1}\boldsymbol{f}_{\boldsymbol{u}}.\end{aligned} \tag{5}$$

Here $\delta\boldsymbol{x}_k := \boldsymbol{x}_k - \bar{\boldsymbol{x}}_k$, $\delta\boldsymbol{u}_k := \boldsymbol{u}_k - \bar{\boldsymbol{u}}_k$ are deviations about the nominal sequences. It is also implied that the Q functions above are evaluated on $\bar{\boldsymbol{X}}$, $\bar{\boldsymbol{U}}$. Plugging (5) into (3), we can explicitly optimize with respect to $\delta\boldsymbol{u}$ and compute the locally optimal control deviations. These will be given by

$$\begin{aligned}&\delta\boldsymbol{u}_k^* = \boldsymbol{k}_k + \boldsymbol{K}_k\delta\boldsymbol{x}_k, \\ \text{with}\quad &\boldsymbol{k} := -Q_{\boldsymbol{uu}}^{-1}Q_{\boldsymbol{u}},\ \ \boldsymbol{K} := -Q_{\boldsymbol{uu}}^{-1}Q_{\boldsymbol{ux}}.\end{aligned} \tag{6}$$

Next, V_k will be quadratically expanded and will be plugged along with $\delta\boldsymbol{u}_k^*$ into (3) to give:

$$\begin{aligned}V_{\boldsymbol{x},k} &= Q_{\boldsymbol{x},k} - Q_{\boldsymbol{xu},k}Q_{\boldsymbol{uu},k}^{-1}Q_{\boldsymbol{u},k} \\ V_{\boldsymbol{xx},k} &= Q_{\boldsymbol{xx},k} - Q_{\boldsymbol{xu},k}Q_{\boldsymbol{uu},k}^{-1}Q_{\boldsymbol{ux},k}.\end{aligned} \tag{7}$$

These equations are propagated backwards in time, as the boundary condition of V is $V(\boldsymbol{x}_N) = \phi(\boldsymbol{x}_N)$ at end of the horizon. After the backward pass is complete, the control sequence is applied to the system in a forward pass, typically via line search to ensure cost reduction, generating the nominal state trajectory for the next iteration. This procedure is repeated until certain convergence criteria are satisfied.

B. Augmented Lagrangian

Consider the optimization problem

$$\min_{\boldsymbol{x}} h(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}, \tag{8}$$

where $\boldsymbol{g} = (g_1(\boldsymbol{x}), \ldots, g_w(\boldsymbol{x}))^\top$ is a vector of w constraints. We define the AL cost function as [17], [18]

$$L_A(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) := h(\boldsymbol{x}) + \textstyle\sum_i \mathscr{P}(g_i(\boldsymbol{x}), \lambda_i, \mu_i), \tag{9}$$

where $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ correspond to the Lagrange multipliers and penalty parameters respectively, while $\mathscr{P}(\cdot)$ is the penalty function for inequalities. When $\mathscr{P}(\cdot)$ satisfies certain properties, it can be shown that the minimization of (9) can give a solution to (8), under mild assumptions [18].

Loosely speaking, the corresponding optimization process can be divided into an inner and outer loop. In the inner loop, a local minimizer is found for (9) by an unconstrained optimization methodology. At the outer loop, the Lagrange multipliers are updated as: $\lambda_i \leftarrow \mathscr{P}'(g_i, \lambda_i, \mu_i)$, where $\mathscr{P}'(y, \lambda, \mu) := \frac{\partial}{\partial y}\mathscr{P}(y, \lambda, \mu)$. Moreover, the penalty parameters are increased monotonically, when constraint improvement is not satisfactory.

The most popular AL algorithm uses the penalty function $\mathscr{P}(y, \lambda, \mu) = \frac{1}{2\mu}(\max(0, \lambda + \mu y)^2 - \lambda^2)$ and is known as the PHR method [17]. Despite its success, one key drawback is that the objective function of each subproblem is not twice differentiable, which may cause numerical instabilities when used within second-order algorithms [18]. We refer the interested reader to [17], [18] for more details about valid penalty functions and their properties.

III. Constrained DDP using KKT conditions and slack variables

We will henceforth focus on the constrained optimal control problem:

$$\begin{aligned}&\min_{\boldsymbol{U}} J(\boldsymbol{X}, \boldsymbol{U}) = \min_{\boldsymbol{U}} \left[\textstyle\sum_{k=0}^{N-1} l(\boldsymbol{x}_k, \boldsymbol{u}_k) + \phi(\boldsymbol{x}_N)\right] \\ &\text{subject to:}\quad \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad g_{i,k}(\boldsymbol{x}_k, \boldsymbol{u}_k) \leq 0, \\ &\qquad\qquad k = 0, ..., N-1, \quad i = 1, ..., w.\end{aligned} \tag{10}$$

Note that we did not include equality constraints above only for compactness. Our results can be readily extended to this case as well.

A. Backward Pass

Similar to normal unconstrained DDP, the backward pass operates on quadratic approximations of the Q functions about the nominal rollouts (see eqs. (3), (4), (5)). For the constrained case, we can write this as:

$$\begin{aligned}&\min_{\delta\boldsymbol{u}_k} Q_k(\delta\boldsymbol{x}_k, \delta\boldsymbol{u}_k) \\ &\text{subject to}\qquad \tilde{\boldsymbol{g}}_k(\bar{\boldsymbol{x}}_k + \delta\boldsymbol{x}_k, \bar{\boldsymbol{u}}_k + \delta\boldsymbol{u}_k) \leq 0.\end{aligned} \tag{11}$$

$\tilde{\boldsymbol{g}}_k$ above is associated with the constraints influenced directly by states and controls at time instance t_k. We will discuss later the selection of such constraints.

We proceed by linearizing the constraints, as well as incorporating the approximate Q function from (5). We have

$$\begin{aligned}&\tilde{\boldsymbol{g}}(\bar{\boldsymbol{x}}_k + \delta\boldsymbol{x}_k, \bar{\boldsymbol{u}}_k + \delta\boldsymbol{u}_k) \\ &\approx \tilde{\boldsymbol{g}}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) + \underbrace{\tilde{\boldsymbol{g}}_{\boldsymbol{u}}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)}_{\boldsymbol{C}_k}\delta\boldsymbol{u}_k + \underbrace{\tilde{\boldsymbol{g}}_{\boldsymbol{x}}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)}_{\boldsymbol{D}_k}\delta\boldsymbol{x}_k.\end{aligned} \tag{12}$$

Now, for the approximate problem of (11), we will apply the Karush-Kuhn-Tucker (KKT) optimality conditions [17]. Letting $\boldsymbol{\lambda}_k$ denote the set of Lagrange multipliers, these will read:

$$Q_{\boldsymbol{uu}}\delta\boldsymbol{u}_k + Q_{\boldsymbol{u}} + Q_{\boldsymbol{ux}}\delta\boldsymbol{x}_k + \boldsymbol{C}^{\mathsf{T}}\boldsymbol{\lambda}_k = \boldsymbol{0} \tag{13}$$

$$\tilde{g}_i \leq 0, \quad \lambda_{i,k} \geq 0, \quad \text{and} \quad \lambda_{i,k}\tilde{g}_i = 0 \tag{14}$$

where we have dropped the time index on the constraints and Q derivatives for simplicity. We rewrite the above conditions by considering slack variables, s_i, such that $s_i + g_i = 0$ and $s_i \geq 0$. Hence, eq. (14) becomes

$$s_{i,k} \geq 0, \quad \lambda_{i,k} \geq 0, \quad s_{i,k}\lambda_{i,k} = 0. \tag{15}$$

To proceed, we will consider perturbations of the slack variables and Lagrange multipliers about their nominal values. Hence we obtain

$$Q_{\boldsymbol{uu}}\delta\boldsymbol{u}_k + Q_{\boldsymbol{u}} + Q_{\boldsymbol{ux}}\delta\boldsymbol{x}_k + \boldsymbol{C}_k^{\mathsf{T}}(\bar{\boldsymbol{\lambda}}_k + \delta\boldsymbol{\lambda}_k) = \boldsymbol{0}, \tag{16}$$

$$(\bar{s}_{i,k} + \delta s_{i,k})(\bar{\lambda}_{i,k} + \delta\lambda_{i,k}) = 0. \tag{17}$$

By omitting the second-order terms, we get

$$\boldsymbol{S}\bar{\boldsymbol{\lambda}} + \boldsymbol{\Lambda}\delta\boldsymbol{s}_k + \boldsymbol{S}\delta\boldsymbol{\lambda}_k = \boldsymbol{0}, \tag{18}$$

where $\boldsymbol{\Lambda} := \mathrm{diag}(\bar{\lambda}_{i,k})$ and $\boldsymbol{S} := \mathrm{diag}(\bar{s}_{i,k})$. Furthermore, the slack formulation of the inequality constraints in (14) will give

$$\boldsymbol{S}\boldsymbol{e} + \delta\boldsymbol{s}_k + \tilde{\boldsymbol{g}}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) + \boldsymbol{C}\delta\boldsymbol{u}_k + \boldsymbol{D}\delta\boldsymbol{x}_k = \boldsymbol{0}, \tag{19}$$

where $\boldsymbol{e} := (1, \ldots, 1)^{\mathsf{T}}$. Overall, the obtained KKT system is,

$$\begin{bmatrix} Q_{\boldsymbol{uu}} & \boldsymbol{0} & \boldsymbol{C}^{\mathsf{T}} \\ \boldsymbol{0} & \boldsymbol{\Lambda} & \boldsymbol{S} \\ \boldsymbol{C} & \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \delta\boldsymbol{u}_k \\ \delta\boldsymbol{s}_k \\ \delta\boldsymbol{\lambda}_k \end{bmatrix} = \begin{bmatrix} -Q_{\boldsymbol{ux}}\delta\boldsymbol{x}_k - Q_{\boldsymbol{u}} - \boldsymbol{C}^{\mathsf{T}}\bar{\boldsymbol{\lambda}}_k \\ -\boldsymbol{S}\bar{\boldsymbol{\lambda}}_k + \mu_k\sigma_k\boldsymbol{e} \\ -\boldsymbol{D}\delta\boldsymbol{x}_k - \tilde{\boldsymbol{g}}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) - \boldsymbol{S}\boldsymbol{e} \end{bmatrix} \tag{20}$$

$$\text{with} \quad s_{i,k} \geq 0, \quad \lambda_{i,k} \geq 0.$$

Here, the duality measure $\mu_k := \bar{\mathbf{s}}_k^{\mathsf{T}}\bar{\boldsymbol{\lambda}}_k/w$ has been introduced to overcome numerical issues when solving the above system [17]. By multiplying by $\sigma_k (0 < \sigma_k < 1)$, this term can be made iteratively smaller, so that, in the limit, the original condition (18) is satisfied.

To solve (20), we will be using the primal-dual interior point method [17]. Specifically, $\bar{\boldsymbol{\lambda}}$ will be initialized as $\bar{\boldsymbol{\lambda}} = \boldsymbol{e}$, while the slack variables will be initialized as $\bar{s}_{k,i} = \max(-g_i, \epsilon)$, where ϵ is a small positive number which helps maintain numerical stability. Now notice that in the backward pass of DDP, we do not have $\delta\boldsymbol{x}$. Hence, our strategy will be to first solve the KKT system on the nominal rollout by substituting $\delta\boldsymbol{x} = \boldsymbol{0}$, and then use our optimal values of $\boldsymbol{s}$ and $\boldsymbol{\lambda}$ as our $\bar{\boldsymbol{s}}$ and $\bar{\boldsymbol{\lambda}}$ for the next KKT iteration.

Finally, using the updated $\bar{\boldsymbol{s}}_k$ and $\bar{\boldsymbol{\lambda}}_k$, we solve the system at the perturbed trajectory ($\delta\boldsymbol{x} \neq 0$) for $\delta\boldsymbol{u}_k$:

$$\delta\boldsymbol{u}_k = -Q_{\boldsymbol{uu}}^{-1}[\boldsymbol{H}Q_u + \boldsymbol{E}^{\mathsf{T}}\boldsymbol{M}\boldsymbol{\Lambda}(\tilde{\boldsymbol{g}}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) + \boldsymbol{S}\boldsymbol{e})] - Q_{\boldsymbol{uu}}^{-1}(\boldsymbol{H}Q_{\boldsymbol{ux}} + \boldsymbol{E}^{\mathsf{T}}\boldsymbol{M}\boldsymbol{\Lambda}\boldsymbol{D})\delta\boldsymbol{x}_k \tag{21}$$

$$\text{with} \quad \boldsymbol{E} := \boldsymbol{\Lambda}\boldsymbol{C}, \quad \boldsymbol{H} := \boldsymbol{I} - \boldsymbol{E}^{\mathsf{T}}\boldsymbol{M}\boldsymbol{E}Q_{\boldsymbol{uu}}^{-1} \tag{22}$$

$$\boldsymbol{F} := \boldsymbol{\Lambda}\boldsymbol{S}, \quad \boldsymbol{M} := (\boldsymbol{E}\boldsymbol{Q}_{uu}^{-1}\boldsymbol{E}^{\mathsf{T}} + \boldsymbol{F})^{-1}. \tag{23}$$

The feedforward gain, $\boldsymbol{k}$, and feedback gain, $\boldsymbol{K}$, can then be obtained as follows:

$$\boldsymbol{k} = -Q_{\boldsymbol{uu}}^{-1}[\boldsymbol{H}\boldsymbol{Q}_u + \boldsymbol{E}^{\mathsf{T}}\boldsymbol{M}\boldsymbol{\Lambda}(\boldsymbol{g}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) + \boldsymbol{S}\boldsymbol{e})] \tag{24}$$

$$\boldsymbol{K} = -Q_{\boldsymbol{uu}}^{-1}(\boldsymbol{H}Q_{\boldsymbol{ux}} + \boldsymbol{E}^{\mathsf{T}}\boldsymbol{M}\boldsymbol{\Lambda}\boldsymbol{D}) \tag{25}$$

We will lastly discuss which constraints, $\tilde{\boldsymbol{g}}$, to consider. We assume the full state $\boldsymbol{x}$ is composed of kinematic states $\boldsymbol{x}^p \in \mathbb{R}^{n_p}$, and dynamic states $\boldsymbol{x}^v \in \mathbb{R}^{n_v}$, where $\boldsymbol{x} = [\boldsymbol{x}^{p\mathsf{T}}, \boldsymbol{x}^{v\mathsf{T}}]^{\mathsf{T}}$ and

$$\boldsymbol{x}_{k+2}^p = \boldsymbol{f}^p(\boldsymbol{x}_{k+1}), \quad \begin{bmatrix} \boldsymbol{x}_{k+1}^p \\ \boldsymbol{x}_{k+1}^v \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}^p(\boldsymbol{x}_k) \\ \boldsymbol{f}^v(\boldsymbol{x}_k, \boldsymbol{u}_k) \end{bmatrix}. \tag{26}$$

This formulation is commonly found in robotics and controls applications. Thus, $\boldsymbol{u}_k$ does not affect $\boldsymbol{x}_k$, but $\boldsymbol{x}_{k+2}^p$, and $\boldsymbol{x}_{k+1}^v$, implying that the QP should be solved subject to constraints two time steps forward for $\boldsymbol{x}^p$, and one time step forward for $\boldsymbol{x}^v$. Hence, $\boldsymbol{C}$ will be generally a non-zero matrix, as opposed to the nominal unconstrained setting.

For most practical applications, the elements of $\boldsymbol{g}_k$ can be divided into separate functions of $\boldsymbol{x}^p$, $\boldsymbol{x}^v$, and $\boldsymbol{u}_k$, which we will denote here as $\boldsymbol{g}_k^p$, $\boldsymbol{g}_k^v$, and $\boldsymbol{g}_k^c$, respectively. For each instance t_k we will have to treat those constraints differently. Specifically, for $\boldsymbol{g}_k^p$ we need to consider two time steps ahead, and for $\boldsymbol{g}_k^v$ one time step ahead, in order to make them explicit functions of $\boldsymbol{u}_k$. By applying the transition dynamics function, the coefficients of linearized $\boldsymbol{g}_{k+2}^p$ can be calculated using the chain rule:

$$\boldsymbol{g}_{k+2}^p(\bar{\boldsymbol{x}}_k + \delta\boldsymbol{x}_k, \bar{\boldsymbol{u}}_k + \delta\boldsymbol{u}_k) \approx \boldsymbol{g}_{k+2}^p(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) + \underbrace{\frac{\partial\boldsymbol{g}_{k+2}^p}{\partial\boldsymbol{x}_k}}_{\boldsymbol{D}_k^p}\delta\boldsymbol{x}_k + \underbrace{\frac{\partial\boldsymbol{g}_{k+2}^p}{\partial\boldsymbol{u}_k}}_{\boldsymbol{C}_k^p}\delta\boldsymbol{u}_k \tag{27}$$

$$\boldsymbol{D}_k^p = \frac{\partial\boldsymbol{g}_{k+2}^p}{\partial\boldsymbol{x}_{k+2}^p}\boldsymbol{f}_{\boldsymbol{x},k+1}^p\boldsymbol{f}_{\boldsymbol{x},k}, \quad \boldsymbol{C}_k^p = \frac{\partial\boldsymbol{g}_{k+2}}{\partial\boldsymbol{x}_{k+2}^p}\boldsymbol{f}_{\boldsymbol{x},k+1}^p\boldsymbol{f}_{\boldsymbol{u},k}. \tag{28}$$

By applying this rule to $\boldsymbol{g}_k^v$, and use $\boldsymbol{g}^c$ for $\boldsymbol{g}_k^c$, we consider the following set of linearized constraints at the kth instance:

$$\tilde{\boldsymbol{g}}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) = \begin{bmatrix} \boldsymbol{g}_{k+2}^p \\ \boldsymbol{g}_{k+1}^v \\ \boldsymbol{g}_k^c \end{bmatrix}, \boldsymbol{C}_k = \begin{bmatrix} \boldsymbol{C}_k^p \\ \boldsymbol{C}_k^v \\ \boldsymbol{C}_k^c \end{bmatrix}, \boldsymbol{D}_k = \begin{bmatrix} \boldsymbol{D}_k^p \\ \boldsymbol{D}_k^v \\ \boldsymbol{D}_k^c \end{bmatrix}. \tag{29}$$

B. Forward Pass

By using the propagated constraints discussed in the previous section, the following Quadratic Programming (QP) problem will be solved to guarantee feasibility:

$$\arg\min_{\delta\boldsymbol{u}_k}[\frac{1}{2}\delta\boldsymbol{u}_k{}^{\mathsf{T}}Q_{\boldsymbol{uu}}\delta\boldsymbol{u}_k + Q_{\boldsymbol{u}}^{\mathsf{T}}\delta\boldsymbol{u}_k + \delta\boldsymbol{u}_k{}^{\mathsf{T}}Q_{\boldsymbol{ux}}\delta\boldsymbol{x}_k] \tag{30}$$

$$\text{subject to} \quad \tilde{\boldsymbol{g}}_k(\bar{\boldsymbol{x}}_k + \delta\boldsymbol{x}_k, \bar{\boldsymbol{u}}_k + \delta\boldsymbol{u}_k) \approx \tilde{\boldsymbol{g}}_k(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) + \boldsymbol{D}_k\delta\boldsymbol{x}_k + \boldsymbol{C}_k\delta\boldsymbol{u}_k \leq \boldsymbol{0}$$

A subtle, but important, difference between the above formulation and the one used in [13] is how $\boldsymbol{C}_k$, $\boldsymbol{D}_k$ are constructed. In the latter, $\tilde{\boldsymbol{g}}_k$ corresponds only to constraints depending on $\boldsymbol{u}_k$, $\boldsymbol{x}_k$, leading to $\boldsymbol{C}_k$, $\boldsymbol{D}_k$ becoming zero

Algorithm 1: Backward Pass

```
1: Initialize: $V_N \leftarrow \phi(\bar{\boldsymbol{x}}_N)$, $V_{\boldsymbol{x},N} \leftarrow \boldsymbol{\nabla}_{\boldsymbol{x}}\phi(\bar{\boldsymbol{x}}_N)$,
   $V_{\boldsymbol{xx},N} \leftarrow \boldsymbol{\nabla}_{\boldsymbol{xx}}\phi(\bar{\boldsymbol{x}}_N)$
2: for $k = N-1$ to 0 do
3:     Calculate $l, Q$, and their derivatives at $k$
4:     Initialize $\bar{s}_{k,i} = \max(-g_i, \epsilon)$, $\bar{\lambda}_{k,i} = 1$
5:     $\mu_0 \leftarrow \boldsymbol{s}^\top \boldsymbol{\lambda}/w$, iter $\leftarrow 0$
6:     while $\mu/\mu_0 < 0.01$ or iter $<$ max_iter do
7:         Solve KKT system to obtain $\delta\boldsymbol{u}_k$, $\delta\boldsymbol{s}_k$, and $\delta\boldsymbol{\lambda}_k$
8:         Update $\bar{\boldsymbol{u}}_k, \bar{\boldsymbol{s}}_k, \bar{\boldsymbol{\lambda}}_k, \mu_k$, iter $\leftarrow$ iter $+ 1$
9:     end while
10:    $\boldsymbol{k} \leftarrow -Q_{\boldsymbol{uu}}^{-1}[\boldsymbol{H}\boldsymbol{Q}_u + \boldsymbol{E}^\top \boldsymbol{M}\boldsymbol{\Lambda}(\boldsymbol{g}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) + \boldsymbol{S}\boldsymbol{e}]$
11:    $\boldsymbol{K} \leftarrow -Q_{\boldsymbol{uu}}^{-1}(\boldsymbol{H}Q_{\boldsymbol{ux}} + \boldsymbol{E}^\top \boldsymbol{M}\boldsymbol{\Lambda}\boldsymbol{D})$
12:    $V_{\boldsymbol{x}} \leftarrow Q_{\boldsymbol{x}} + \boldsymbol{K}^\top Q_{\boldsymbol{uu}}\boldsymbol{k} + \boldsymbol{K}^\top Q_{\boldsymbol{u}} + Q_{\boldsymbol{ux}}^\top \boldsymbol{k}$
13:    $V_{\boldsymbol{xx}} \leftarrow Q_{\boldsymbol{xx}} + \boldsymbol{K}^\top Q_{\boldsymbol{uu}}\boldsymbol{K} + \boldsymbol{K}^\top Q_{\boldsymbol{ux}} + Q_{\boldsymbol{ux}}^\top \boldsymbol{K}$
14: end for
15: Store Derivatives of $Q$
```

Algorithm 2: Forward Pass

```
1: Calculate $J_{\text{int}} \leftarrow J(\boldsymbol{X}, \boldsymbol{U})$
2: $\boldsymbol{x} \leftarrow \boldsymbol{x}_0$
3: $\boldsymbol{X}_{\text{temp}} \leftarrow \boldsymbol{X}$,  $\boldsymbol{U}_{\text{temp}} \leftarrow \boldsymbol{U}$
4: for $k = 0$ to $N-1$ do
5:     $\delta\boldsymbol{x} \leftarrow \boldsymbol{x} - \boldsymbol{x}_k$
6:     $\boldsymbol{x}_{\text{temp},k} \leftarrow \boldsymbol{x}$
7:     $\delta\boldsymbol{u}_k^* =$
       $\arg\min[\frac{1}{2}\delta\boldsymbol{u}_k^\top Q_{\boldsymbol{uu}}\delta\boldsymbol{u}_k + \delta\boldsymbol{u}_k^\top(Q_{\boldsymbol{u}} + Q_{\boldsymbol{ux}}\delta\boldsymbol{x}_k)]$,
       subject to $\boldsymbol{g}_k(\boldsymbol{x}, \boldsymbol{u}_k) + \boldsymbol{C}_k(\boldsymbol{x}, \boldsymbol{u}_k)\delta\boldsymbol{u}_k \leq \boldsymbol{0}$
8:     $\boldsymbol{u}_{\text{temp},k} = \boldsymbol{u}_k + \delta\boldsymbol{u}_k^*$
9:     $\boldsymbol{x} \leftarrow \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}_{\text{temp},k})$
10: end for
11: Calculate $J_{\text{temp}} \leftarrow J(\boldsymbol{X}_{\text{temp}}, \boldsymbol{U}_{\text{temp}})$
12: if $J_{\text{temp}} < J_{\text{int}}$ then
13:     $\boldsymbol{X} \leftarrow \boldsymbol{X}_{\text{temp}}, \boldsymbol{U} \leftarrow \boldsymbol{U}_{\text{temp}}$
14:     Decrease $\nu_1$ and $\nu_2$.
15: else
16:     Increase $\nu_1$ and $\nu_2$.
17: end if
```

for many problems. In this setting, solving (30) may not result in feasible trajectories. The authors in [13] suggest also using a trust region on the control updates $\delta\boldsymbol{u}_k$ when feasibility cannot be attained, but this methodology will rarely be beneficial when the initial sequence fed to the algorithm violates the constraints. Also, this method makes the algorithm susceptible to initial trajectory since the trajectory gets conservative to satisfy constraints by the trust region.

C. Regularization

In DDP, regularization plays an important role and highly affects its convergence [5]. We use the regularization scheme and scheduling technique proposed in [19] with $\nu_1, \nu_2 > 0$,

$$Q_{\boldsymbol{xx}} = \boldsymbol{l}_{\boldsymbol{xx}} + \boldsymbol{f}_{\boldsymbol{x}}^\top(V'_{\boldsymbol{xx}} + \nu_1 \boldsymbol{I}_n)\boldsymbol{f}_{\boldsymbol{x}} \tag{31}$$

$$Q_{\boldsymbol{ux}} = \boldsymbol{l}_{\boldsymbol{ux}} + \boldsymbol{f}_{\boldsymbol{u}}^\top(V'_{\boldsymbol{xx}} + \nu_1 \boldsymbol{I}_n)\boldsymbol{f}_{\boldsymbol{x}} \tag{32}$$

$$Q_{\boldsymbol{uu}} = \boldsymbol{l}_{\boldsymbol{uu}} + \boldsymbol{f}_{\boldsymbol{u}}^\top(V'_{\boldsymbol{xx}} + \nu_1 \boldsymbol{I}_n)\boldsymbol{B} + \nu_2 \boldsymbol{I}_m \tag{33}$$

These regularized derivatives are used to calculate gains $\boldsymbol{k}$ and $\boldsymbol{K}$, as shown in Algorithm 1.

IV. Constrained DDP with Augmented Lagrangian

A. AL-DDP and penalty function

We will now develop an alternative trajectory optimization scheme for solving (10), based on the Augmented Lagrangian approach. We shall refer to this technique as AL-DDP. The main idea is to observe that *partial elimination of constraints* can be used on the inequality constraints $\boldsymbol{g}$ of (10). This means that the penalty function $\mathscr{P}$ from Section II-B

$$\min_{\boldsymbol{U}} \underbrace{\Big[J(\boldsymbol{X}, \boldsymbol{U}) + \sum_{i=1}^{w} \sum_{k=0}^{N-1} \mathscr{P}(\lambda_i^k, \mu_i^k, g_{i,k}(\boldsymbol{x}_k, \boldsymbol{u}_k)) \Big]}_{L_A}$$

subject to $\quad \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k),\ k = 0, \cdots N-1.$

Based on the discussion of Section II-B, we will be using the standard unconstrained DDP method (see Section II-A) to optimize (34), followed by an update on the multipliers and the penalty parameters. Since DDP requires $L_A(\cdot)$ to be twice differentiable, $\mathscr{P}$ is selected as

$$\mathscr{P}(\lambda_{i,k}, \mu_{i,k}, g_{i,k}(\boldsymbol{x}_k)) = \frac{(\lambda_{i,k})^2}{\mu_{i,k}} \phi\left(\frac{\mu_{i,k}}{\lambda_{i,k}} g_{i,k}(\boldsymbol{x}_k)\right) \tag{34}$$

$$\phi(t) := \begin{cases} \frac{1}{2}t^2 + t, & t \geq -\frac{1}{2} \\ -\frac{1}{4}\log(-2t) - \frac{3}{8}, & \text{otherwise,} \end{cases} \tag{35}$$

which can be seen as a smooth approximation to PHR [18].

B. Combination of AL-DDP and KKT frameworks

The AL approach is typically robust to initializations of the algorithm, but displays slow convergence rates near the (local) solution [18]. To tackle this drawback, we combine the two approaches discussed in our paper: We first use the AL-DDP formulation until a pre-specified precision of the cost and constraints, and subsequently switch to the KKT-based approach of Section III. When the cost reduction after the switch is not sufficient, or the trajectory fed to the KKT is not good enough to obtain feasibility, we switch back to the AL method, and reduce the "switching" tolerances.

Another reason that justifies our switching strategy is the following: The KKT-based method can handle hard constraints explicitly, but, from our experience, may usually lead to a situation where state and control constraints conflict with each other; that is when the QP in (30) becomes infeasible. This may happen because the forward pass of the algorithm takes only two time steps forward into account, and becomes more noticeable for applications with tight control bounds. Conversely, we will show in our examples that using

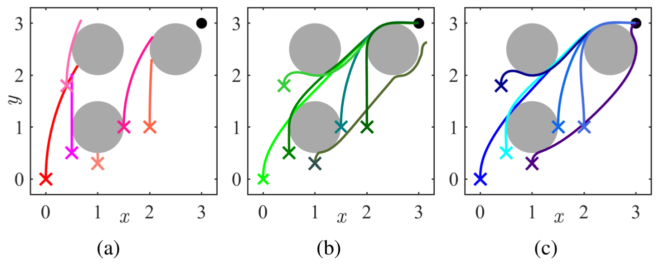

Fig. 1. Obtained car trajectories starting from six different initial points, marked as "×". The mean cost for each algorithm at the final iteration is (a) CDDP [13]: $\bar{J}$ =356, (b) Active set: $\bar{J}$ =5.88, (c) S-KKT, $\bar{J}$ =2.91. Feasibility was attained for all of them.

a (possibly slightly infeasible) trajectory from AL-DDP as a warm start to our KKT-DDP framework, typically helps avoid conflicts in the forward pass of the latter.

V. Results

Here we provide simulation results and comparisons between our method and prior work. We will henceforth refer to the methods developed in sections V-A and IV as "S-KKT" and "AL-DDP", respectively, and the combination thereof (see section IV-B) as "AL-S-KKT".

To exemplify the effect of the slack variables-based formulation, we also test a slightly different version of S-KKT, where the active set method is used instead of slack variables, as is done in [14]. More precisely, constraints are taken into account in the backward pass only when they are close to being active, assuming also that they will remain active at the next iteration. We will refer to this method as "Active set" in our graphs. Comparisons will also be made against the previous method in [13], termed here "CDDP".

A. S-KKT

1) 2D car: We use the 2D car model as described in [13]. The system's state vector is $\boldsymbol{x} = [x, y, \theta, v]^\mathsf{T}$ indicating its position, steering angle and velocity, while the controls inputs are $\boldsymbol{u} = [u^\theta, u^v]$, indicating their effects on the steering angle and velocity, respectively. The task we consider is reaching a target point at $\boldsymbol{x}_g = [3, 3, \frac{\pi}{2}, 0]^\mathsf{T}$ while avoiding three circular obstacles. We design the cost by setting $l(\boldsymbol{x}_k, \boldsymbol{u}_k) = \boldsymbol{u}_k^\mathsf{T} \boldsymbol{R} \boldsymbol{u}_k$ and $\phi(\boldsymbol{x}_N) = (\boldsymbol{x}_k - \boldsymbol{x}_g)^\mathsf{T} \boldsymbol{A}_\mathrm{f} (\boldsymbol{x}_k - \boldsymbol{x}_g)$ in (10), where $\boldsymbol{R}$ and $\boldsymbol{A}_\mathrm{f}$ are positive semidefinite matrices.

Fig. 1 shows the results of each algorithm starting statically from six different initial locations. S-KKT achieves the smoothest trajectories with the lowest cost. Fig. 2 shows the outputs of each methodology, for three different initial trajectories. We observe that even when S-KKT is initialized with an infeasible trajectory, it can still get out of the prohibited area.

2) Quadrotor: We test our algorithm on a quadrotor system, defined in [13], [20]. The state $\boldsymbol{x} \in \mathbb{R}^{12}$ consists of three Euler angles and positions, and their velocities. The control input $\boldsymbol{u} \in \mathbb{R}^4$ is the force generated by the rotors. The task for the quadrotor is to reach a goal position $\boldsymbol{x}_g = [1, 5, 5]^\mathsf{T}$ avoiding three obstacles. Fig. 3 shows the obtained trajectories for each algorithm starting from four

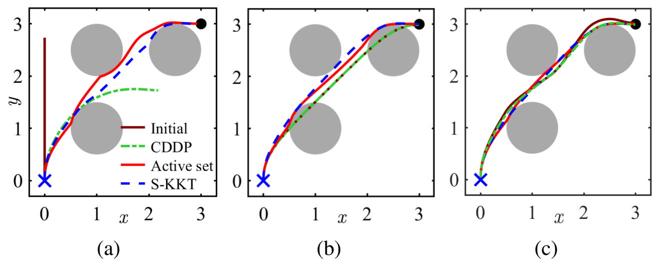

Fig. 2. Obtained car trajectories with three different initial sequences, termed $\overrightarrow{\boldsymbol{x}_0}$. The mean cost and feasibility for each method are: (a) $\overrightarrow{\boldsymbol{x}_0}$ feasible: i) CDDP: $\bar{J}$ = 127-feasible, ii) Active set: $\bar{J}$ = 4.96-feasible, iii) S-KKT: $\bar{J}$ = 4.90-feasible, b) $\overrightarrow{\boldsymbol{x}_0}$ infeasible: i) CDDP: $\bar{J}$ =3.11-infeasible, ii) Active set: $\bar{J}$ = 3.42-feasible, iii) S-KKT: $\bar{J}$ = 3.27-feasible, (c) $\overrightarrow{\boldsymbol{x}_0}$ close to optimal: i) CDDP: $\bar{J}$ =3.92-feasible, ii) Active set: $\bar{J}$ = 3.45-feasible, iii) S-KKT: $\bar{J}$ = 3.30-feasible.

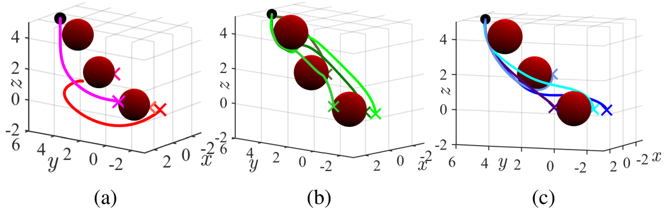

Fig. 3. Trajectories from different algorithms. Starting hovering points are shown by "×". (a) CDDP: $\bar{J}$ = 2093, (b) Active set: $\bar{J}$ = 2.40, and (c) S-KKT: $\bar{J}$ = 0.699. Feasibility was attained for all of them.

different hovering points. S-KKT outperforms the remaining methods in this scenario as well.

B. Combination of S-KKT and AL-DDP

We show here the results of our combination scheme AL-S-KKT, emphasizing on the effect of S-KKT after the switch. We also consider additional control constraints

1) 2D car: We consider the same problem setting as in Section V-A.1, with additional control limit: $u^\theta \in [-\frac{\pi}{3}, \frac{\pi}{3}]$ and $u^v \in [-6, 6]$. As shown in Fig. 4, the algorithm can successfully handle both state and (tight) control constraints.

2) Quadrotor: We consider the same setting as in V-A.2, with additional control limits $\boldsymbol{u} \in [0, \frac{mg_0}{2}]$, where the parameters of mass: m and gravitational acceleration: g_0 are given in [20]. The results are shown in Fig. 5, where the vertical dotted line corresponds to the iteration when S-KKT starts. Despite this being an infeasible trajectory, S-KKT still achieves feasibility with a slight increase the in the cost.

C. Performance Analysis

Next we perform an analysis between five algorithms, SQP, S-KKT-DDP, AL-DDP, AL-DDP with SQP, and AL-DDP with S-KKT. In the five algorithms, the last two, AL-DDP with SQP and AL-DDP with S-KKT are a combination of two different optimization algorithms. In these two combination methods, the algorithms start optimizing using AL-DDP and switch to SQP or S-KKT as explained in IV-B. We compare the algorithms under a time budget in terms of cost, time, and feasibility metrics for the 2D car and quadrotor used in V-A. Different time horizons are tested, but the time step size dt is fixed in each example. In 2D car $dt = 0.02$

TABLE I

COMPARISON OF DIFFERENT METHODS.

Example	Metric	SQP	S-KKT	AL-DDP	AL-SQP	AL-S-KKT
2D car	Cost	25.3	22.5	3.13	2.49	**2.44**
Time budget = 3s	Time	3	2.77	3	3	**2.36**
N = 100	Feasibility	$\boldsymbol{f}$: 1.023E-03	0	0	$\boldsymbol{f}$: 2.53E-05	0
2D car	Cost	135	43.2	**1.71**	1.80	1.92
Time budget = 6s	Time	6	6	6	6	**3.88**
N = 200	Feasibility	$\boldsymbol{f}$: 1.22E-02	0	$\boldsymbol{g}$: 1.73E-03	$\boldsymbol{f}$: 1.10E-05	0
Quadrotor	Cost	2.43E03	26.1	10.7	8.12	**7.48**
Time budget = 6s	Time	6	**5.68**	6	6	5.91
N = 200	Feasibility	$\boldsymbol{f}$: 1.86E-03	0	0	$\boldsymbol{f}$: 7.99E-04	0
Quadrotor	Cost	2.86E03	**7.52**	7.91	8.01	7.78
Time budget = 10s	Time	10	10	10	10	10
N = 300	Feasibility	$\boldsymbol{f}$: 2.50E-03	0	0	$\boldsymbol{f}$: 6.85E-09	0

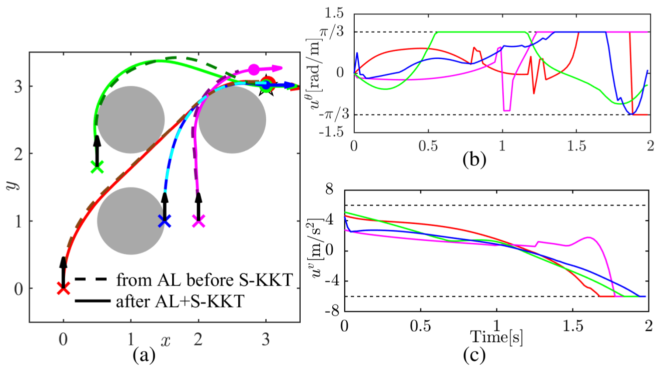

Fig. 4. Car reaching task. (a) Trajectories, (b) Steering control, (c) Acceleration control. Mean cost after AL and before S-KKT: $\bar{J}$ = 11.7, After S-KKT: $\bar{J}$ = 4.38. Feasibility was attained for all trajectories.

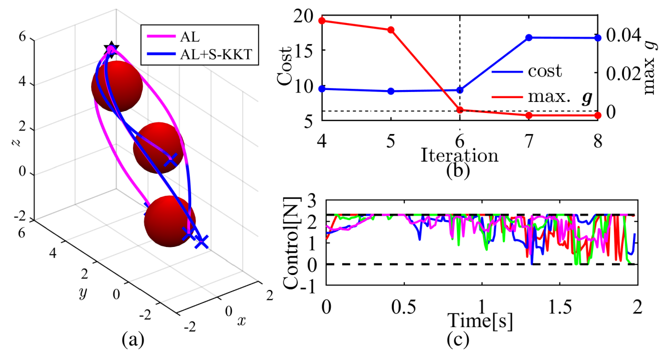

Fig. 5. Quadrotor reaching task. (a) trajectory, (b) iteration, cost and constraint of one trajectory. (c) control of the same trajectory shown used in (b). Note: AL and AL-S-KKT visually overlap in (a). Mean cost after AL and before S-KKT, $\bar{J}$ = 7.22 with constraint violation = 0.0016, After S-KKT, $\bar{J}$ = 9.08 and feasibility is attained.

and quadrotor $dt = 0.01$. The feasibility of the solution is divided into two parts, the constraints $\boldsymbol{g}$ and the dynamics $\boldsymbol{f}$. SQP handles them as inequalities (for $\boldsymbol{g}$) and equalities (for $\boldsymbol{f}$) constraints, whereas DDP-based methods implicitly satisfy dynamics since they are used during the optimization process.

1) Exit criteria: The exit condition of optimization was based on two criteria. One was constraint satisfaction, where 1×10^{-7} was used for all the algorithms. The other was an optimality criterion. For the DDP based methods, we used the change in the optimization objective between iterations, set to 8×10^{-2}. For SQP, we used an optimality condition by choosing the Matlab `fmincon` option `OptimalityTolerance` to be 1×10^{-2}. For AL-SQP and AL-S-KKT, we also have conditions on exiting the AL optimization scheme and switching to the next scheme. The constraint satisfaction tolerance was 1×10^{-2} and the bound on the cost change was 1.

2) Performance: Table V-B.2 shows the results of the comparison. In the row of Feasibility, the largest violation of the two constraints, dynamics: $\boldsymbol{f}$ and constraints: $\boldsymbol{g}$ are presented. If both of them are satisfied, 0 is filled. Pure SQP obtained the highest cost with dynamics violation, while requiring a much longer time for a longer time horizon compared to other methods. This is because in SQP, a longer time horizon corresponds to a larger matrix (which needs to be inverted) containing the equality constraints for the dynamics. Combining with AL-DDP, it performed much better in terms of cost, but still had a violation in dynamics. We also observed that running until convergence, SQP could achieve the lowest cost but constraints are not strictly satisfied. We notice the performance of the AL-DDP and S-KKT combination in terms of speed and constraint satisfaction.

VI. CONCLUSION

In this paper we have introduced novel constrained trajectory optimization methods that outperform previous versions of constrained DDP. In particular for S-KKT, by introducing slack variables and forward expansion, our method is robust to initial conditions, and achieves a lower cost. This method, however, can not handle a situation when a large control, exceeding the control bound, is required to satisfy the state constraints. To solve this problem, we combine S-KKT and AL-DDP which can strictly satisfy control constraints, but has difficulty satisfying state constraints. With the combination, the weakness for both is compensated and a better algorithm which handles both control and state constraints is obtained. Future directions will include mechanisms for uncertainty representations and learning, as well as development of chance constrained trajectory optimization algorithms that have the benefits of the proposed algorithms' fast convergence.

References

[1] D. H. Jacobson and D. Q. Mayne, *Differential dynamic programming*. Elsevier, 1970.

[2] V. Kumar, E. Todorov, and S. Levine, “Optimal control with learned local models: Application to dexterous manipulation,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 378–383.

[3] J. Morimoto, G. Zeglin, and C. G. Atkeson, “Minimax differential dynamic programming: Application to a biped walking robot,” in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 2. IEEE, 2003, pp. 1927–1932.

[4] S. Levine and P. Abbeel, “Learning neural network policies with guided policy search under unknown dynamics,” in *Advances in Neural Information Processing Systems*, 2014, pp. 1071–1079.

[5] L.-Z. Liao and C. A. Shoemaker, “Convergence in unconstrained discrete-time differential dynamic programming,” *IEEE Transactions on Automatic Control*, vol. 36, no. 6, pp. 692–706, 1991.

[6] W. Li and E. Todorov, “Iterative linear quadratic regulator design for nonlinear biological movement systems.” in *ICINCO (1)*, 2004, pp. 222–229.

[7] M. B. Kobilarov and J. E. Marsden, “Discrete geometric optimal control on lie groups,” *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 641–655, 2011.

[8] K. Mombaur, “Using optimization to create self-stable human-like running,” *Robotica*, vol. 27, no. 3, pp. 321–330, 2009.

[9] Y. Tassa, N. Mansard, and E. Todorov, “Control-limited differential dynamic programming,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1168–1175.

[10] D. M. Murray and S. J. Yakowitz, “Constrained differential dynamic programming and its application to multireservoir control,” *Water Resources Research*, vol. 15, no. 5, pp. 1017–1027, 1979.

[11] M. Giftthaler and J. Buchli, “A projection approach to equality constrained iterative linear quadratic optimal control,” in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, 2017, pp. 61–66.

[12] G. Lantoine and R. Russell, “A hybrid differential dynamic programming algorithm for robust low-thrust optimization,” in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, 2008, p. 6615.

[13] Z. Xie, C. K. Liu, and K. Hauser, “Differential dynamic programming with nonlinear constraints,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 695–702.

[14] T. Lin and J. Arora, “Differential dynamic programming technique for constrained optimal control,” *Computational Mechanics*, vol. 9, pp. 27–40, 1991.

[15] B. Plancher, Z. Manchester, and S. Kuindersma, “Constrained unscented dynamic programming,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 5674–5680.

[16] T. A. Howell, B. E. Jackson, and Z. Manchester, “Altro: A fast solver for constrained trajectory optimization,” in *2019 IEEE International Conference on Intelligent Robots and Systems, IEEE*, 2019.

[17] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[18] E. G. Birgin, R. A. Castillo, and J. M. Martínez, “Numerical comparison of augmented lagrangian algorithms for nonconvex problems,” *Computational Optimization and Applications*, vol. 31, no. 1, pp. 31–55, 2005.

[19] Y. Tassa, T. Erez, and E. Todorov, “Synthesis and stabilization of complex behaviors through online trajectory optimization,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 4906–4913.

[20] T. Luukkonen, “Modelling and control of quadcopter,” *Independent research project in applied mathematics, Espoo*, 2011.