ALTRO: A Fast Solver for Constrained Trajectory Optimization

Taylor A. Howell[1*], Brian E. Jackson[1*], and Zachary Manchester[2]

***Abstract*— Trajectory optimization is a widely used tool for robot motion planning and control. Existing solvers for these problems either rely on off-the-shelf nonlinear programming solvers that are numerically robust and capable of handling arbitrary constraints, but tend to be slow because they are general purpose; or they use custom numerical methods that take advantage of the problem structure to be fast, but often lack robustness and have limited or no ability to reason about constraints. This paper presents ALTRO (Augmented Lagrangian TRajectory Optimizer), a solver for constrained trajectory optimization problems that handles general nonlinear state and input constraints and offers fast convergence and numerical robustness thanks to careful exploitation of problem structure. We demonstrate its performance on a set of benchmark motion-planning problems and offer comparisons to the standard direct collocation method with large-scale sequential quadratic programming and interior-point solvers.**

I. Introduction

Trajectory optimization is a powerful tool for motion planning, enabling the synthesis of dynamic motion for complex underactuated robotic systems. This general framework can be applied to robots with nonlinear dynamics and constraints where other motion planning paradigms—such as sample-based planning, inverse dynamics, or differential flatness—might be ineffective or impractical.

Numerical trajectory optimization algorithms solve variations of the following problem,

$$\begin{aligned} \underset{x_{0:N},\, u_{0:N-1},\, \Delta t}{\text{minimize}} \quad & \ell_N(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k, \Delta t) \\ \text{subject to} \quad & x_{k+1} = f(x_k, u_k, \Delta t), \\ & g_k(x_k, u_k) \leq 0, \\ & h_k(x_k, u_k) = 0, \end{aligned} \tag{1}$$

where k is the time-step index, ℓ_N and ℓ_k are the terminal and stage costs, x_k and u_k are the states and control inputs, Δt is the duration of a time step, $f(x_k, u_k, \Delta t)$ is the discrete dynamics, and $g_k(x_k, u_k)$ and $h_k(x_k, u_k)$ are inequality and equality constraints.

To solve such problems, "direct" methods treat both states and controls as decision variables and use general-purpose nonlinear programming (NLP) solvers, such as SNOPT [1] and IPOPT [2], that tend to be versatile and robust. It is straight forward to provide an initial state trajectory to the solver with such methods, even if the trajectory is dynamically infeasible. Direct transcription (DIRTRAN) [3] and direct collocation (DIRCOL) [4] are common direct methods.

Alternatively, "indirect" methods exploit the Markov structure of (1) and only treat the control inputs as decision variables, with the dynamics constraints implicitly enforced by simulating the system's dynamics. Differential Dynamic Programming (DDP) [5] and iterative LQR (iLQR) [6] are closely related indirect methods that solve (1) by breaking it into a sequence of smaller sub-problems. Because of their strict enforcement of dynamic feasibility, it is often difficult to find a control sequence that produces a reasonable initialization for DDP methods. While they are fast and have a low memory footprint, making them amenable to embedded implementation, DDP methods have historically been considered less numerically robust and less well-suited to handling nonlinear state and input constraints.

Several efforts have been made to incorporate constraints into DDP methods: box constraints on controls [7] and stage-wise inequality constraints on the states [8], [9] have been handled by solving a constrained quadratic program (QP) at each sub-problem step. A projection method was also devised that satisfies linearized terminal state and stage state-control constraints [10]. Augmented Lagrangian (AL) methods have been proposed [11], including hybrid approaches that also solve constrained QPs for stage state-control constraints [9], [12]. Mixed state-control constraints have also been handled using a penalty method [13]. Unfortunately, all of these methods either have limitations on the types of constraints they can handle, or suffer from numerical instability and conditioning issues.

This paper presents ALTRO (Augmented Lagrangian TRajectory Optimizer), a solver for constrained trajectory optimization problems of the form (1), that combines the best characteristics of direct and DDP methods, namely: speed, numerical robustness, handling of general state and input constraints, and initialization with infeasible state trajectories. ALTRO combines iLQR with an augmented Lagrangian method to handle general state and input constraints and an active-set projection method for final "solution polishing" to achieve fast and robust overall convergence. Our novel contributions are: 1) a numerically robust square-root formulation of the DDP algorithm, 2) a method for initializing DDP with an infeasible state trajectory, 3) a strategy for solving minimum-time problems with DDP, and 4) an objective-weighted Newton projection method for solution polishing.

The remainder of the paper is organized as follows: Section II reviews the iLQR algorithm with augmented Lagrangian constraint handling, which we denote AL-iLQR, along with some results on matrix square roots. Section

[1]Department of Mechanical Engineering, Stanford University, USA {thowell,bjack205}@stanford.edu

[2]Department of Aeronautics and Astronautics, Stanford University, USA zacmanchester@stanford.edu

*These authors contributed equally to this work

III presents the ALTRO algorithm in detail. Comparisons between ALTRO and DIRCOL are then provided for several motion-planning problems in Section IV. Finally, we summarize our findings in Section V.

II. Background

Notation: For a function $\ell(x,u) : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}$, we define $\ell_x \equiv \partial \ell / \partial x \in \mathbf{R}^n$, $\ell_{xx} \equiv \partial^2 \ell / \partial x^2 \in \mathbf{R}^{n \times n}$, and $\ell_{xu} \equiv \partial^2 \ell / \partial x \partial u \in \mathbf{R}^{n \times m}$. We also use the notation (A, B, C) to indicate the vertical concatenation of matrices, $[A^T B^T C^T]^T$.

A. Augmented Lagrangian iLQR

We present a succinct description of AL-iLQR comprising two parts: the modified iLQR iteration, and the Lagrange multiplier and penalty updates used between successive iLQR steps.

1) Modified iLQR: From (1), the augmented Lagrangian is:

$$\begin{aligned}
&\mathcal{L}_A(X, U, \lambda, \mu) = \ell_N(x_N) \\
&+ (\lambda_N + \frac{1}{2} I_{\mu_N} c_N(x_N))^T c_N(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k, \Delta t) \\
&+ (\lambda_k + \frac{1}{2} I_{\mu_k} c_k(x_k, u_k))^T c_k(x_k, u_k) \\
&= \mathcal{L}_N(x_N, \lambda_N, \mu_N) + \sum_{k=0}^{N-1} \mathcal{L}_k(x_k, u_k, \lambda_k, \mu_k)
\end{aligned} \tag{2}$$

where $\lambda_k \in \mathbf{R}^{p_k}$ is a Lagrange multiplier, $\mu_k \in \mathbf{R}^{p_k}$ is a penalty weight, and $c_k = (g_k, h_k) \in \mathbf{R}^{p_k}$ is the concatenated set of inequality and equality constraints with index sets $\mathcal{I}_k$ and $\mathcal{E}_k$, respectively. I_{μ_k} is a diagonal matrix defined as,

$$I_{\mu_k, ii} = \begin{cases} 0 & \text{if } c_{k_i}(x_k, u_k) < 0 \wedge \lambda_{k_i} = 0,\ i \in \mathcal{I} \\ \mu_{k_i} & \text{otherwise,} \end{cases} \tag{3}$$

where k_i indicates the ith constraint at time step k.

The dynamics constraints are handled implicitly using an initial state x_0 and nominal control trajectory, $U = \{u_0, \ldots, u_{N-1}\}$, to simulate forward the state trajectory $X = \{x_0 \ldots, x_N\}$.

The backward pass is derived by defining the optimal cost-to-go for fixed multipliers and penalty terms, $V|_{\lambda,\mu}$, and the recurrence relationship,

$$\begin{aligned}
V_N(x_N)|_{\lambda,\mu} &= \mathcal{L}_N(x_N, \lambda_N, \mu_N) \\
V_k(x_k)|_{\lambda,\mu} &= \min_{u_k} \{\mathcal{L}_k(x_k, u_k, \lambda_k, \mu_k) \\
&\quad + V_{k+1}(f(x_k, u_k))|_{\lambda,\mu}\} \\
&= \min_{u_k} Q(x_k, u_k)|_{\lambda,\mu},
\end{aligned}$$

where $Q_k = Q(x_k, u_k)|_{\lambda,\mu}$ is the action-value function. To make the dynamic programming step tractable, we take a second-order Taylor expansion of V_k with respect to the state variable and fixed λ and μ,

$$\delta V_k \approx p_k^T \delta x_k + \frac{1}{2} \delta x_k^T P_k \delta x_k, \tag{4}$$

resulting in the optimal terminal cost-to-go second-order expansion,

$$\begin{aligned}
p_N &= (\ell_N)_x + (c_N)_x^T (\lambda + I_{\mu_N} c_N) && (5) \\
P_N &= (\ell_N)_{xx} + (c_N)_x^T I_{\mu_N} (c_N)_x. && (6)
\end{aligned}$$

The relationship between δV_k and δV_{k+1} is derived by taking the second-order Taylor expansion of Q_k with respect to the state and control,

$$\delta Q_k = \frac{1}{2} \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix}^T \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix} + \begin{bmatrix} Q_x \\ Q_u \end{bmatrix}^T \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix} \tag{7}$$

Dropping the time-step indices for notational clarity, the block matrices are,

$$\begin{aligned}
Q_{xx} &= \ell_{xx} + A^T P' A + c_x^T I_\mu c_x && (8) \\
Q_{uu} &= \ell_{uu} + B^T P' B + c_u^T I_\mu c_u && (9) \\
Q_{ux} &= \ell_{ux} + B^T P' A + c_u^T I_\mu c_x && (10) \\
Q_x &= \ell_x + A^T p' + c_x^T (\lambda + I_\mu c) && (11) \\
Q_u &= \ell_u + B^T p' + c_u^T (\lambda + I_\mu c), && (12)
\end{aligned}$$

where $A = \partial f / \partial x|_{x_k, u_k}$, $B = \partial f / \partial u|_{x_k, u_k}$, and $'$ indicates variables at the $k+1$ time step. Consistent with the use of linearized dynamics in iLQR, linearized constraints are also used in the expansion.

Minimizing (7) with respect to δu_k gives a correction to the control trajectory. The result is a feedforward term d_k and a linear feedback term $K_k \delta x_k$. Regularization is added to ensure the invertibility of Q_{uu}:

$$\delta u_k^* = -(Q_{uu} + \rho I)^{-1} (Q_{ux} \delta x_k + Q_u) \equiv K_k \delta x_k + d_k. \tag{13}$$

Substituting δu_k^* back into (7), a closed-form expression for p_k, P_k, and the expected change in cost, ΔV_k, is found:

$$\begin{aligned}
P_k &= Q_{xx} + K_k^T Q_{uu} K_k + K_k^T Q_{ux} + Q_{xu} K_k && (14) \\
p_k &= Q_x + K_k^T Q_{uu} d_k + K_k^T Q_u + Q_{xu} d_k && (15) \\
\Delta V_k &= d_k^T Q_u + \frac{1}{2} d_k^T Q_{uu} d_k. && (16)
\end{aligned}$$

A forward pass then simulates the system using the correction to the nominal control trajectory and a line search is performed on the feedforward term d_k to ensure a reduction in cost.

2) Augmented Lagrangian Updates: After an iLQR step with λ and μ held constant, the dual variables are updated according to,

$$\lambda_{k_i}^+ = \begin{cases} \lambda_{k_i} + \mu_{k_i} c_{k_i}(x_k^*, u_k^*) & i \in \mathcal{E}_k \\ \max(0, \lambda_{k_i} + \mu_{k_i} c_{k_i}(x_k^*, u_k^*)) & i \in \mathcal{I}_k, \end{cases} \tag{17}$$

and the penalty is increased monotonically according to the schedule,

$$\mu_{k_i}^+ = \phi \mu_{k_i}, \tag{18}$$

where $\phi > 1$ is a scaling factor. The solve-update cycle is then repeated until a desired optimality and constraint tolerances are achieved. DDP variants have previously been used to solve the inner minimization of the AL method with good results [9], [11].

Algorithm 1 AL-iLQR

```
1: function AL-iLQR($x_0, U, \text{tol.}$)
2:     Initialize $\lambda, \mu, \phi$
3:     while $\max(c) >$ tol. do
4:         minimize $\mathcal{L}_A(X, U, \lambda, \mu)$ using iLQR
5:         Update $\lambda$ using (17), update $\mu$ using (18)
6:     end while
7: return $X, U, \lambda$
8: end function
```

B. *Matrix Square Roots*

The QR decomposition $F = QR$ returns an upper triangular matrix R and orthogonal matrix Q. The Cholesky decomposition $G = U^T U$ of a positive-definite matrix G returns an upper-triangular "square-root" matrix U, which we denote $U = \sqrt{G}$. For A, $B \in \mathbf{R}^{n\times n}$, we use these factorizations to calculate $\sqrt{A+B}$ from $\sqrt{A}$ and $\sqrt{B}$. Note that $A + B = \begin{bmatrix} \sqrt{A}^T & \sqrt{B}^T \end{bmatrix} (\sqrt{A}, \sqrt{B}) = F^T F$, where $F \in \mathbf{R}^{2n\times n}$. Using the QR decomposition $F = QR$, $A + B = R^T Q^T Q R = R^T R$, which gives us a Cholesky factor $R = \sqrt{A+B}$. We define $\sqrt{A+B} \leftarrow \text{QR}\Big((\sqrt{A}, \sqrt{B})\Big)$. Similarly, $\sqrt{A-B}$ can be computed by performing successive rank-one downdates on $\sqrt{A}$ using the rows of $\sqrt{B}$. We define $\sqrt{A-B} \leftarrow \text{DD}(\sqrt{A}, \sqrt{B})$.

III. ALTRO

ALTRO (Algorithm 4) comprises two stages: The first stage solves (1) rapidly to a coarse tolerance using a version of AL-iLQR with several novel refinements. The second stage uses the coarse solution from the first stage to warm start an active-set projection method that achieves high-precision constraint satisfaction. We now present our extensions to AL-iLQR, as well as the details of the projection method.

A. *Square-Root Backward Pass*

For AL methods to achieve fast convergence, the penalty terms must be increased to large values, which can result in severe numerical ill-conditioning [14]. To help mitigate this issue and make ALTRO more numerically robust, especially on embedded processors with limited numerical precision, we introduce a square-root backward pass inspired by the square-root Kalman filter [15]. cholesky分解前提：矩阵是PD

We now derive recursive expressions for the following upper-triangular Cholesky factors: $S \equiv \sqrt{P}, Z_{xx} \equiv \sqrt{Q_{xx}}$, and $Z_{uu} \equiv \sqrt{Q_{uu}}$, using the methods from section II-B. The square root of the terminal cost-to-go Hessian (6) is,

$$S_N \leftarrow \text{QR}\Big(\big(\sqrt{(\ell_N)_{xx}}, \sqrt{I_{\mu_N}}(c_N)_x\big)\Big), \tag{19}$$

and the action-value expansion factorizations, (8) and (9), are,

$$Z_{xx} \leftarrow \text{QR}\Big(\big(\sqrt{\ell_{xx}}, S' A, \sqrt{I_\mu} c_x\big)\Big) \tag{20}$$

$$Z_{uu} \leftarrow \text{QR}\Big(\big(\sqrt{\ell_{uu}}, S' B, \sqrt{I_\mu} c_u, \sqrt{\rho} I\big)\Big), \tag{21}$$

The gains K and d from (13) are expressed in square-root form as,

$$K = -Z_{uu}^{-1} Z_{uu}^{-T} Q_{ux} \tag{22}$$

$$d = -Z_{uu}^{-1} Z_{uu}^{-T} Q_u, \tag{23}$$

and the gradient (15) and expected change (16) of the cost-to-go are,

$$p = Q_x + (Z_{uu}K)^T (Z_{uu}d) + K^T Q_u + Q_{xu} d \tag{24}$$

$$\Delta V = d^T Q_u + \frac{1}{2}(Z_{uu}d)^T (Z_{uu}d). \tag{25}$$

The square root of the the cost-to-go Hessian (14)—which frequently exhibits the worst numerical conditioning—is derived by assuming the following upper-triangular Cholesky factorization,

$$\begin{aligned} P &= \begin{bmatrix} I \\ K \end{bmatrix}^T \begin{bmatrix} Z_{xx}^T & 0 \\ C^T & D^T \end{bmatrix} \begin{bmatrix} Z_{xx} & C \\ 0 & D \end{bmatrix} \begin{bmatrix} I \\ K \end{bmatrix} \\ &= \begin{bmatrix} Z_{xx} + CK \\ DK \end{bmatrix}^T \begin{bmatrix} Z_{xx} + CK \\ DK \end{bmatrix} \end{aligned} \tag{26}$$

where,

$$C = Z_{xx}^{-T} Q_{xu} \tag{27}$$

$$D = \sqrt{Z_{uu}^T Z_{uu} - Q_{ux} Z_{xx}^{-1} Z_{xx}^{-T} Q_{xu}}. \tag{28}$$

S can then be computed with a QR decomposition:

$$S \leftarrow \text{QR}\left(\begin{bmatrix} Z_{xx} + Z_{xx}^{-T} Q_{xu} K \\ \text{DD}(Z_{uu}, Z_{xx}^{-T} Q_{xu}) K \end{bmatrix} \right). \tag{29}$$

B. *Infeasible State Trajectory Initialization*

Desired state trajectories can often be identified (e.g., from sampling-based planners or expert knowledge), whereas finding a control trajectory that will produce this result is usually challenging. Dynamically infeasible state trajectory initialization is enabled by introducing additional inputs to the dynamics with slack controls $s \in \mathbf{R}^n$,

$$x_{k+1} = f(x_k, u_k) + s_k, \tag{30}$$

to make the system artificially fully actuated.

Given initial state and control trajectories, $\tilde{X}$ and U, the initial infeasible controls $s_{0:N-1}$ are computed as the difference between the desired state trajectory and the dynamics at each time step:

$$s_k = \tilde{x}_{k+1} - f(\tilde{x}_k, u_k) \tag{31}$$

The optimization problem (1) is modified by replacing the dynamics with (30). An additional cost term,

$$\sum_{k=0}^{N-1} \frac{1}{2} s_k^T R_s s_k, \tag{32}$$

and constraints $s_k = 0,\ k{=}0,\dots,N{-}1$ are also added to the problem. Since $s_k = 0$ at convergence, a dynamically feasible solution to (1) is still obtained.

C. Time-Penalized Problems

Minimum-time, or more general time-penalized problems, can be solved by considering $\tau_k = \sqrt{\Delta t_k} \in \mathbf{R}$ as an input at each time step. To ensure the solver does not exploit discretization errors in the system dynamics, equality constraints must be added between time step durations. The optimization problem (1) is modified to use dynamics,

$$\begin{bmatrix} x_{k+1} \\ \delta_{k+1} \end{bmatrix} = \begin{bmatrix} f(x_k, u_k, \tau_k) \\ \tau_k \end{bmatrix}, \tag{33}$$

with an additional cost term,

$$\sum_{k=0}^{N-1} R_\tau \tau_k^2, \tag{34}$$

subject to $\delta_k = \tau_k,\ k{=}1,\dots,N{-}1$ constraints, and upper and lower bounds on τ_k.

Algorithm 2 Projection

```
1:  function PROJECTION($Y$, tol.)
2:      $H^{-1} \leftarrow$ invert Hessian of objective
3:      while $\|d\|_\infty >$ tol. do
4:          $d, D \leftarrow$ linearize active constraints
5:          $S \leftarrow \sqrt{DH^{-1}D^T}$
6:          $v \leftarrow \|d\|_\infty$
7:          $r \leftarrow \infty$
8:          while $v >$ tol. and $r >$ conv. rate tol. do
9:              $Y, v^+ \leftarrow$ LINESEARCH($Y, S, H^{-1}, D, d, v$)
10:             $r \leftarrow \log v^+ / \log v$
11:         end while
12:     end while
13: return $Y$
14: end function
```

Algorithm 3 Projection Line Search

```
1:  function LINESEARCH($Y, S, H^{-1}, D, d, v_0$)
2:      Initialize $\alpha, \gamma$
3:      while $v > v_0$ do
4:          $\delta Y_p \leftarrow H^{-1} D^T (S^{-1} S^{-T} d)$
5:          $Y_p \leftarrow Y_p + \alpha \delta Y_p$
6:          $d \leftarrow$ UPDATECONSTRAINTS($\bar{Y}_p$)
7:          $v \leftarrow \|d\|_\infty$
8:          $\alpha \leftarrow \gamma \alpha$
9:      end while
10: return $\bar{Y}, v$
11: end function
```

D. Active-Set Projection Method

The coarse primal and dual trajectories, $Y \leftarrow X, U, \lambda$, returned from the AL-iLQR stage of ALTRO are used to warm start an active-set projection method (Algorithm 2). This hybrid approach avoids the numerical ill-conditioning and slow tail convergence exhibited by AL methods when the penalty weights are made large. To ensure strict satisfaction of the dynamics and constraints, the current solution is projected onto the manifold associated with the active constraints d. The norm used to compute distances from this manifold is weighted by the Hessian of the objective H. Algorithm 2 takes successive Newton steps δY, only updating the constraint Jacobian D when the convergence rate r drops below a threshold, allowing re-use of the same matrix factorization S for inexpensive linear system solutions. Further, this algorithm can be implemented in a sequential manner [16] that does not require building large matrices, making it amenable to embedded systems.

Algorithm 4 ALTRO

```
1:  procedure
2:      Initialize $x_0, U$, tolerances; $\tilde{X}$
3:      if Infeasible Start then
4:          $X \leftarrow \tilde{X}$, $s_{0:N-1} \leftarrow$ from (31)
5:      else
6:          $X \leftarrow$ Simulate from $x_0$ using $U$
7:      end if
8:      $X, U, \lambda \leftarrow$ AL-iLQR($X, U$, tol.)
9:      $(X, U, \lambda) \leftarrow$ PROJECTION($(X, U, \lambda)$, tol.)
10: return $X, U$
11: end procedure
```

IV. Simulation Results

ALTRO's timing and constraint-satisfaction performance is compared to DIRCOL [4] on a number of benchmark motion-planning problems. Each problem uses a quadratic objective, has initial and terminal state constraints, is solved to constraint satisfaction $c_{\max} = 1e\text{-}8$ ($c_{\max} = 1e\text{-}6$ for time-penalized problems), and is performed on a desktop computer with an AMD Ryzen Threadripper 2950x processor and 16GB RAM. ALTRO is implemented purely in the Julia programming language, while DIRCOL uses the Ipopt and SNOPT NLP solvers through a Julia interface. DIRCOL is provided the dynamically feasible state trajectory computed during the initial forward simulation from ALTRO. Further details regarding the problems and parameters used, along with full source code, is available on GitHub.

A. Problem Descriptions

1) *1D Block Move:* Double Integrator with two states and one input. The system is tasked to move one unit length subject to control limits.

2) *Pendulum:* The system must swing from the downward equilibrium to the upward equilibrium point subject to control limits.

3) *Acrobot:* Double pendulum system limited to actuation at the "elbow" joint. Tasked with swinging upright from the downward position.

4) *Cartpole:* The system must perform a swing-up maneuver while obeying control limits.

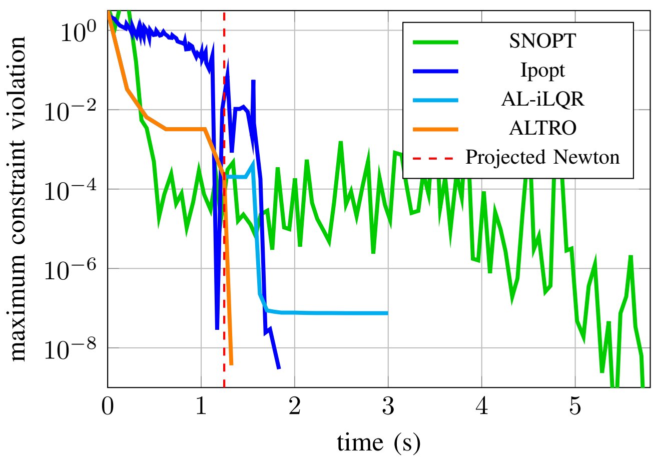

Fig. 1. Maximum constraint violation comparison for the cartpole. After solving to a coarse tolerance, ALTRO performs a single iteration of Algorithm 2 and converges to the specified constraint tolerance. AL-iLQR fails to converge to the specified tolerance without the projection step.

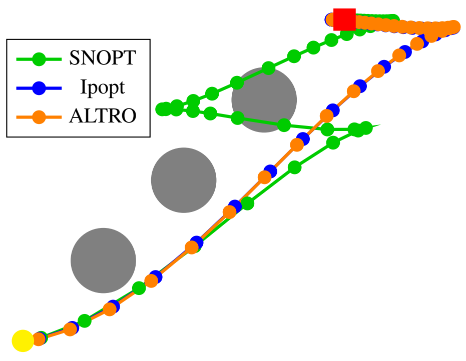

Fig. 2. Trajectories for the Reeds-Shepp car with obstacles. ALTRO and Ipopt converge to similar trajectories, while SNOPT finds a different solution that exploits discretization error. Yellow and red markers are start and goal positions, respectively.

5) Reeds-Shepp Car: A differential-drive system with three states and two controls that can move forward or backward [17] performs a few different tasks: *Parallel Park*: Move one unit in the direction perpendicular to the direction of motion (i.e., "sideways") and end in the same orientation while obeying state and control constraints. Also performed with time penalization. *Obstacles*: Move to goal while avoiding 3 circular obstacles (Figure 2). *Escape*: Shown in Figure 3, the car must move from one "room" to another while obeying control limits. For this task, solvers are initialized with a state trajectory interpolated from 6 2D positions.

6) Robot Arm: A serial manipulator with 14 states and 7 inputs, modeled after the Kuka iiwa, is tasked with moving from an initial to final configuration through an environment with obstacles and subject to torque limits (Figure 4). The control trajectory is initialized to compensate for gravity.

7) Quadrotor: The system with 13 states (quaternion angular representation) and 4 inputs is tasked with navigating through a maze with floor and ceiling constraints (Figure 5). Inputs are bounded and initialized to perform hovering. For the maze task, solvers are initialized with a state trajectory interpolated from 7 3D positions.

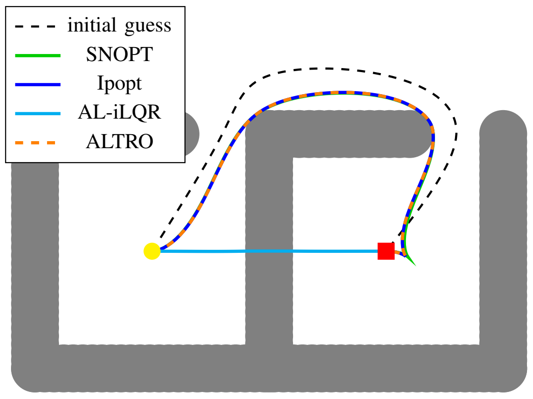

Fig. 3. 2D position trajectories for car escape problem. ALTRO, Ipopt, and SNOPT converge to the same solution, but AL-iLQR fails to find a collision-free trajectory. Yellow and red markers indicate start and goal positions, respectively.

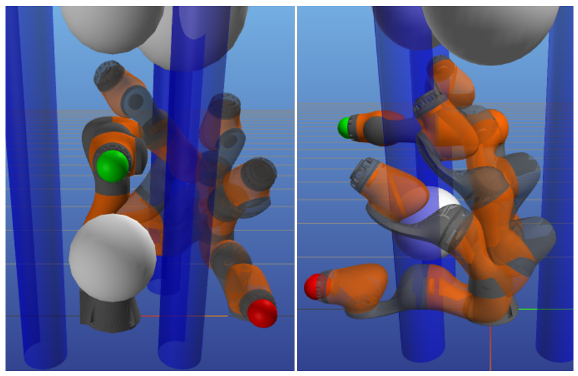

Fig. 4. Front (left) and side (right) views of the Kuka iiwa arm moving its end effector from an initial position (red) to a desired position (green) while avoid obstacles.

Timing results for the benchmark problems are included in Table I. ALTRO is 2-5 times faster than both SNOPT and Ipopt for most problems, except for time-penalized problems, where DIRCOL performs considerably better than ALTRO. Constraint satisfaction versus solver time is shown for the cartpole problem in Figure 1. Infeasible state trajectory initialization is demonstrated on the Car Escape and Quadrotor Maze problems where AL-iLQR fails to find collision free trajectories. DIRCOL was unable to find feasible solutions for the Robot Arm and Quadrotor Maze obstacle avoidance problems.

V. Discussion

ALTRO performs competitively in terms of both computation time and constraint satisfaction when compared to DIRCOL on a variety of benchmark problems. ALTRO's rapid convergence on constraint satisfaction demonstrated in Figure 1 is typical of all the benchmark problems. By using an AL method to make rapid initial progress, then switching to an active-set method once the active inequality constraints are known, fast convergence can be achieved throughout the entire solve process.

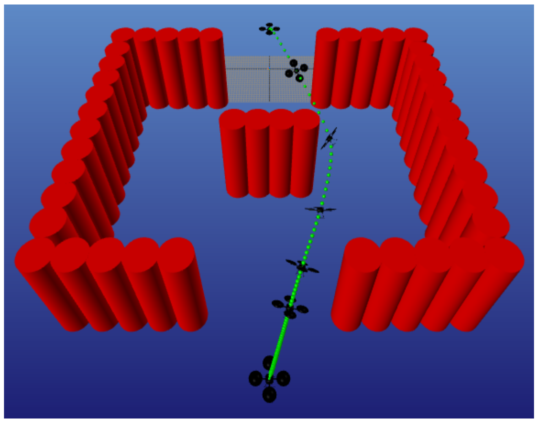

Fig. 5. Quadrotor navigating maze environment.

TABLE I
RUNTIME PEFORMANCE: ALTRO VS DIRCOL

System	ALTRO	Ipopt	SNOPT
Block Move	**12 ms**	26 ms	31 ms
Pendulum	**65 ms**	213 ms	646 ms
Pendulum Time Pen.	848 ms	**134 ms**	225 ms
Acrobot	**1.2 s**	11.9 s	6.1 s
Cartpole	**661 ms**	1.1 s	3.8 s
Parallel Park	**63 ms**	138 ms	385 ms
Parallel Park Time Pen.	5.0 s	175 ms	**119 ms**
Car w/ 3 Obs.	**136 ms**	935 ms	2.4 s
Car Escape	**1.2 s**	9.2 s	37.2 s
Kuka w/ Obs.	9.2 s	-	-
Quadrotor Line	**2.3 s**	4.9 s	7.9 s
Quadrotor Maze	33.0 s	-	-

For time-penalized problems, DIRCOL finds lower total trajectory times and solves faster and more reliably compared to ALTRO. This may be due to inherent sensitivity to initial time steps in shooting methods like iLQR.

One of ALTRO's most significant advantages over AL-iLQR and other DDP methods is the ability to be initialized with infeasible state trajectories. Even though the dimension of the input vector is increased, ALTRO is able to solve these problems reliably and faster than DIRCOL.

DIRCOL was unable to find feasible solutions for two of the obstacle avoidance problems. This is likely due to the way constraints were represented in simulation as well as the differences in the way constraints are handled between the penalty, interior point, and active-set methods used by ALTRO, Ipopt, and SNOPT, respectively.

Future improvements to ALTRO include parallelization of constraint and Jacobian evaluations, cost expansions, and dynamics simulations, while more significant parallelization may be achieved using Alternating Direction Method of Multipliers techniques, a natural extension to the AL method used in ALTRO. The sequential version of Algorithm 2 will make ALTRO more amenable to implementation on embedded hardware. Future work will investigate using ALTRO to solve bi-level optimization problems including contact-implicit trajectory optimization and Stackelberg competitions.

In conclusion, we have presented a new solver for constrained trajectory optimization problems, ALTRO, that combines the advantages of direct and DDP methods. Compared to a standard tool for robot motion planning, DIRCOL, ALTRO exhibits comparable capabilities and typically faster solve times on a variety of benchmark problems. Our implementation of ALTRO is available at `https://github.com/RoboticExplorationLab/TrajectoryOptimization.jl`.

ACKNOWLEDGEMENTS

This work was supported by a NASA Early Career Faculty Award (Grant Number 80NSSC18K1503). This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1656518. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP Algorithm for Large-scale Constrained Optimization," *SIAM Review*, vol. 47, no. 1, pp. 99–131, 2005.

[2] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," en, *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, Mar. 2006.

[3] D. Pardo, L. Möller, M. Neunert, A. W. Winkler, and J. Buchli, "Evaluating direct transcription and nonlinear optimization methods for robot motion planning," pp. 1–9, Apr. 2015.

[4] C. R. Hargraves and S. W. Paris, "Direct Trajectory Optimization Using Nonlinear Programming and Collocation," *J. Guidance*, vol. 10, no. 4, pp. 338–342, 1987.

[5] D. Q. Mayne, "A second-order gradient method of optimizing non- linear discrete time systems," *Int J Control*, vol. 3, p. 8595, 1966.

[6] W. Li and E. Todorov, "Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems," in *Proceedings of the 1st International Conference on Informatics in Control, Automation and Robotics*, Setubal, Portugal, 2004.

[7] Y. Tassa, T. Erez, and E. Todorov, "Control-Limited Differential Dynamic Programming," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, May 2014.

[8] Z. Xie, C. K. Liu, and K. Hauser, "Differential dynamic programming with nonlinear constraints," en, in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, Singapore: IEEE, May 2017, pp. 695–702.

[9] T. C. Lin and J. S. Arora, "Differential dynamic programming technique for constrained optimal control," en, *Computational Mechanics*, vol. 9, no. 1, pp. 27–40, Jan. 1991.

[10] M. Giftthaler and J. Buchli, "A Projection Approach to Equality Constrained Iterative Linear Quadratic Optimal Control," en, *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pp. 61–66, Nov. 2017.

[11] B. Plancher, Z. Manchester, and S. Kuindersma, "Constrained Unscented Dynamic Programming," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC, 2017.

[12] G. Lantoine and R. P. Russell, "A Hybrid Differential Dynamic Programming Algorithm for Constrained Optimal Control Problems. Part 1: Theory," en, *Journal of Optimization Theory and Applications*, vol. 154, no. 2, pp. 382–417, Aug. 2012.

[13] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," en, in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, Singapore: IEEE, May 2017, pp. 93–100.

[14] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, 1996.

[15] J. Bellantoni and K. Dodge, "A square root formulation of the Kalman-Schmidt filter," *AIAA journal*, vol. 5, no. 7, pp. 1309–1314, 1967.

[16] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of Interior-Point Methods to Model Predictive Control," en, *Journal of Optimization Theory and Applications*, vol. 99, no. 3, pp. 723–757, Dec. 1998.

[17] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," en, *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, Oct. 1990.