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 with many important applications in robotic motion planning and control. However, most existing algorithm implementations fall into one of two categories: either they rely on general-purpose off-the-shelf nonlinear programming solvers that are numerically robust and capable of handling arbitrary constraints but tend to be slow, or they use custom numerical methods that are fast but lack robustness and have limited or no ability to deal with constraints. This paper presents ALTRO (Augmented Lagrangian TRajectory Optimizer), a novel algorithm for solving constrained trajectory optimization problems that bridges this gap by offering fast convergence, numerical robustness, and the ability to handle general nonlinear state and input constraints. We demonstrate ALTRO's capabilities on a set of benchmark motion-planning problems and offer comparisons to the standard direct collocation (DIRCOL) method.**

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—are impractical or ineffective.

Numerical trajectory optimization algorithms all solve some variation of the following optimization problem,

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

where ℓ_f and ℓ are the final and stage cost functions, x_k and u_k are the state and input control variables, $f(x_k, u_k)$ is the discrete dynamics function, and $g_k(x_k, u_k)$ and $h_k(x_k, u_k)$ are the inequality and equality constraints, potentially including initial and final conditions, at time step k.

"Direct" methods transcribe states and inputs as decision variables and solve (1) using general-purpose nonlinear programming (NLP) solvers such as SNOPT [1] or Ipopt [2], and tend to be versatile and robust. It is straight forward to provide an initial state trajectory to the solver in such methods, even if it is dynamically infeasible. Direct transcription (DIRTRAN) [3] and direct collocation (DIRCOL) [4] are common direct algorithms.

[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

Alternatively, "indirect" methods leverage the structure of (1) to solve a sequence of smaller sub-problems using Dynamic Programming. These are anytime algorithms, meaning they are always strictly dynamically feasible, allowing state and input trajectories at any iteration to be used in a tracking controller. However, it is often difficult to find a suitable initial guess for the control trajectory. Historically, indirect methods have been considered less robust and less suitable for reasoning about general state and control constraints but tend to be fast and amenable to implementation in embedded systems. Methods include Differential Dynamic Programming (DDP) [5] and Iterative LQR (iLQR) [6], as well as various shooting methods [7].

Several efforts have been made to incorporate constraints into DDP methods: Box constraints on control inputs [8] and stage-wise inequality constraints on the states [9], [10] have been handled by solving a constrained quadratic program (QP) at each step of the backward pass (BP). A projection method was devised that satisfies linearized terminal state and stage state-input constraints [11]. Augmented Lagrangian (AL) methods have been proposed [12], including hybrid approaches that also solve constrained QPs for stage state-input constraints [10], [13]. Mixed state-input constraints have also been handled using a penalty method [14].

In this paper we present ALTRO (Augmented Lagrangian TRajectory Optimizer), a trajectory optimization algorithm that combines the best characteristics of both direct and indirect methods, namely: speed, small problem size, numerical robustness, handling of general state and input constraints, anytime dynamic feasibility, and infeasible state trajectory initialization. Using iLQR in an augmented Lagrangian framework to handle general state and input constraints, we: 1) derive a numerically robust square-root formulation of the backward pass, 2) introduce a method for initializing an infeasible state trajectory, 3) formulate the minimum time problem, and 4) present an anytime projected Newton method for solution polishing.

The paper is organized as follows: Section II provides background for iLQR and AL methods. Section III derives the constrained trajectory optimization algorithm ALTRO. Comparisons between ALTRO and DIRCOL are performed for several motion-planning problems in Section IV. Finally, we summarize our findings in Section V.

II. Background

Notation: For a function $f(x, u)$, we define $f_x \equiv \partial f(x,u)/\partial x|_{x_k,u_k}$, $f_{xx} \equiv \partial^2 f(x,u)/\partial x^2|_{x_k,u_k}$, and $f_{xu} \equiv \partial^2 f(x,u)/\partial x \partial u|_{x_k,u_k}$. We also define a vertical concatenation, $(A, B, C) \equiv [A^T B^T C^T]^T$.

A. Iterative LQR

Iterative LQR (Algorithm 1) minimizes a general cost function,

$$J(X,U) = \ell_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k), \tag{2}$$

subject to the nonlinear dynamics,

$$x_{k+1} = f(x_k, u_k), \tag{3}$$

where $x \in \mathbf{R}^n$ is the system state, and $u \in \mathbf{R}^m$ is a control input. 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 the state trajectory $X = \{x_0 \ldots, x_N\}$.

The backward pass (BP) (Algorithm 2) is derived by defining the optimal cost-to-go, $V_k(x)$, with the recurrence relationship,

$$V_N(x_N) = \ell_f(x_N) \tag{4}$$

$$V_k(x_k) = \min_{u_k}\{\ell(x_k, u_k) + V_{k+1}(f(x_k, u_k))\} \tag{5}$$

$$= \min_{u_k} Q_k(x_k, u_k), \tag{6}$$

where $Q_k(x_k, u_k)$ is the action-value function. To make the dynamic programming step tractable, we take a second-order Taylor expansion of $V_k(x_k)$,

$$\delta V_k(x_k) \approx p_k^T \delta x_k + \frac{1}{2}\delta x_k^T P_k \delta x_k, \tag{7}$$

and linearize the dynamics, resulting in the optimal terminal cost-to-go,

$$p_N = \partial \ell_f(x)/\partial x|_{x_N} \tag{8}$$

$$P_N = \partial^2 \ell_f(x)/\partial x^2|_{x_N}. \tag{9}$$

The relationship between p_k, P_k and p_{k+1}, P_{k+1} is derived by expanding Q_k,

$$\delta Q_k = \frac{1}{2}\begin{bmatrix}\delta x_k \\ \delta u_k\end{bmatrix}^T \begin{bmatrix}Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu}\end{bmatrix}\begin{bmatrix}\delta x_k \\ \delta u_k\end{bmatrix} + \begin{bmatrix}Q_x \\ Q_u\end{bmatrix}^T \begin{bmatrix}\delta x_k \\ \delta u_k\end{bmatrix}, \tag{10}$$

where the block matrices are,

$$Q_{xx} = \ell_{xx} + A_k^T P_{k+1} A_k \tag{11}$$

$$Q_{uu} = \ell_{uu} + B_k^T P_{k+1} B_k \tag{12}$$

$$Q_{ux} = \ell_{ux} + B_k^T P_{k+1} A_k \tag{13}$$

$$Q_x = \ell_x + A_k^T p_{k+1} \tag{14}$$

$$Q_u = \ell_u + B_k^T p_{k+1}, \tag{15}$$

and $A_k = \partial f(x,u)/\partial x|_{x_k,u_k}$, $B_k = \partial f(x,u)/\partial u|_{x_k,u_k}$.

Minimizing (10) 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},

$$\delta u_k^* = -(Q_{uu} + \rho I)^{-1}(Q_{ux}\delta x_k + Q_u) \equiv K_k \delta x_k + d_k. \tag{16}$$

Substituting δu_k^* back into (10), a closed-form expression for p_k, P_k, and the expected change in cost ΔV_k is found,

$$p_k = Q_x + K_k^T Q_{uu} d_k + K_k^T Q_u + Q_{xu} d_k \tag{17}$$

$$P_k = Q_{xx} + K_k^T Q_{uu} K_k + K_k^T Q_{ux} + Q_{xu} K_k \tag{18}$$

$$\Delta V_k = d_k^T Q_u + \frac{1}{2} d_k^T Q_{uu} d_k. \tag{19}$$

A forward pass (FP) (Algorithm 3) simulates the system using the correction to the nominal control trajectory. A line search is performed on the feedforward term d_k to ensure a reduction in cost.

Algorithm 1 Iterative LQR

```
1: Initialize x_0, U, tolerance
2: X ← Simulate from x_0 using U, (3)
3: function iLQR(X, U)
4:     J ← Using X, U, (2)
5:     do
6:         J⁻ ← J
7:         K, d, ΔV ← BackwardPass(X, U)
8:         X, U, J ← ForwardPass(X, U, K, d, ΔV, J⁻)
9:     while |J − J⁻| > tolerance
10: return X, U, J
11: end function
```

Algorithm 2 Backward Pass

```
1: function BackwardPass(X, U)
2:     p_N, P_N ← (8), (9)
3:     for k=N-1:-1:0 do
4:         δQ ← (10), (11)-(15)
5:         if Q_uu ≻ 0 then
6:             K, d, ΔV ← (16), (19)
7:         else
8:             Increase ρ and go to line 3
9:         end if
10:    end for
11: return K, d, ΔV
12: end function
```

B. Augmented Lagrangian

Augmented Lagrangian methods (Algorithm 4) minimize a constrained optimization problem,

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & c_{\mathcal{I}}(x) \leq 0, \\ & c_{\mathcal{E}}(x) = 0 \end{aligned} \tag{20}$$

where $\mathcal{I}$ and $\mathcal{E}$ are the index sets corresponding to the inequality and equality constraints, respectively. AL methods solve (20) by converting it to an unconstrained optimization problem and adding a penalty function to the Lagrangian,

$$\mathcal{L}_A(x, \lambda, \mu) = f(x) + \lambda^T c(x) + \frac{1}{2} c(x)^T I_\mu c(x), \tag{21}$$

Algorithm 3 Forward Pass

function FORWARDPASS($X, U, K, d, \Delta V, J$)
  Initialize $\bar{x}_0 = x_0$, $\alpha = 1$, $J^- \leftarrow J$
  for k=0:1:N-1 **do**
    $\bar{u}_k = u_k + K_k(\bar{x}_k - x_k) + \alpha d_k$
    $\bar{x}_{k+1} \leftarrow$ Using $\bar{x}_k, \bar{u}_k$, (3)
  end for
  $J \leftarrow$ Using X, U, (2)
  if J satisfies line search conditions **then**
    $X \leftarrow \bar{X}, U \leftarrow \bar{U}$
  else
    Reduce α and go to line 3
  end if
return X, U, J
end function

Algorithm 4 Augmented Lagrangian

function AULA(x_0, SOLVER, tolerance)
  Initialize λ, μ, ϕ
  while $\max(c) >$ tolerance **do**
    Minimize $\mathcal{L}_A(x, \lambda, \mu)$ w.r.t. x using SOLVER
    Update λ using (23), update μ using (24)
  end while
return X, λ
end function

where I_μ is a diagonal matrix defined as,

$$I_{\mu,ii} = \begin{cases} 0 & \text{if } c_i(x) < 0 \wedge \lambda_i = 0, \text{ for } i \in \mathcal{I} \\ \mu_i & \text{otherwise.} \end{cases} \tag{22}$$

Holding λ and μ constant, an approximate minimizer of the unconstrained optimization problem, $\hat{x}^*$, is found. Taking the gradient of (21) with respect x suggests the following dual update,

$$\lambda_{i+1} = \begin{cases} \lambda_i + \mu_i c_i(\hat{x}^*) & i \in \mathcal{E} \\ \max(0, \lambda_i + \mu_i c_i(\hat{x}^*)) & i \in \mathcal{I}, \end{cases} \tag{23}$$

while the penalty is increased monotonically according to the schedule,

$$\mu_{i+1} = \phi_i \mu_i, \tag{24}$$

where $\phi_i > 1$ is a scaling factor. The dual and penalty variables are updated and the process is repeated until a desired constraint tolerance is achieved. AL methods make rapid initial progress, but suffer from slow constraint convergence once the penalty is capped at a maximum finite value [15]. DDP has previously been used to solve the inner minimization of the AL method with good results [10], [12].

III. THE ALTRO ALGORITHM

ALTRO (Algorithm 6) comprises two stages: The first stage solves (1) rapidly to a coarse tolerance using iLQR to solve the unconstrained sub-problems within the AL framework, similar to [12]. The optional secondary stage uses the coarse solution from the first stage to warm start an active-set Newton method that achieves high-precision constraint satisfaction. We present several refinements and extensions to constrained iLQR, as well as the novel projected Newton stage for "solution polishing."

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 ill-conditioning. To help mitigate this issue, we introduce a numerically robust BP inspired by the square-root Kalman filter [16].

We derive recursive expressions for the following upper-triangular Cholesky factor matrices: $S \equiv \sqrt{P}, Z_{xx} \equiv \sqrt{Q_{xx}}$, and $Z_{uu} \equiv \sqrt{Q_{uu}}$. The terminal cost-to-go Hessian is,

$$S_N \leftarrow \text{QR}\Big((\sqrt{\partial^2 \ell_f(x)/\partial x^2|_{x_N}}, \sqrt{I_{\mu_N}} c_{x_N}) \Big), \tag{25}$$

and the action-value expansion factorizations explicitly are,

$$Z_{xx} \leftarrow \text{QR}\Big((\sqrt{\ell_{xx}}, S_{k+1} A_k, \sqrt{I_\mu} c_x) \Big) \tag{26}$$

$$Z_{uu} \leftarrow \text{QR}\Big((\sqrt{\ell_{uu}}, S_{k+1} B_k, \sqrt{I_\mu} c_u, \sqrt{\rho} I) \Big), \tag{27}$$

where the 3rd and 4th terms come from taking the gradient of the AL and adding regularization, respectively. The function $\text{QR}(\cdot)$ returns the upper triangular factor of a QR factorization (i.e. R). The gains K and d are expressed in square root form as,

$$K_k = -Z_{uu}^{-1} Z_{uu}^{-T} Q_{ux} \tag{28}$$

$$d_k = -Z_{uu}^{-1} Z_{uu}^{-T} Q_u, \tag{29}$$

and the gradient and expected change of the cost-to-go are,

$$p_k = Q_x + (Z_{uu} K_k)^T (Z_{uu} d_k) + K_k^T Q_u + Q_{xu} d_k \tag{30}$$

$$\Delta V_k = d_k^T Q_u + \frac{1}{2} (Z_{uu} d_k)^T (Z_{uu} d_k). \tag{31}$$

The square root of the the cost-to-go Hessian (18)—which frequently exhibits the worst numerical conditioning—is derived by assuming the following upper triangular Cholesky factorization,

$$\begin{aligned} P &= \begin{bmatrix} I \\ K \end{bmatrix}^T \begin{bmatrix} Z_{xx}^T & 0 \\ C^T & D^T \end{bmatrix} \begin{bmatrix} Z_{xx} & C \\ 0 & D \end{bmatrix} \begin{bmatrix} I \\ K \end{bmatrix} \\ &= \begin{bmatrix} Zxx + CK \\ DK \end{bmatrix}^T \begin{bmatrix} Z_{xx} + CK \\ DK \end{bmatrix} \end{aligned} \tag{32}$$

where,

$$C = Zxx^{-T} Q_{ux} \tag{33}$$

$$D = \sqrt{Z_{uu}^T Z_{uu} - C^T C}. \tag{34}$$

$S = \sqrt{P}$ can then be computed with a QR decomposition:

$$S \leftarrow \text{QR}\Big(\begin{bmatrix} Z_{xx} + CK \\ DK \end{bmatrix} \Big). \tag{35}$$

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 a desired state trajectory is often challenging. State trajectory initialization is enabled by augmenting the discrete dynamics with "infeasible" controls $w \in \mathbf{R}^n$,

$$x_{k+1} = f(x_k, u_k) + w_k, \tag{36}$$

to make the system artificially fully actuated.

Given initial state and control trajectories, $\tilde{X}$ and U, the initial infeasible controls $W = \{w_0, \dots, w_{N-1}\}$ are computed as the difference between the dynamics and desired state trajectory at each time step:

$$w_k = \tilde{x}_{k+1} - f(x_k, u_k) \tag{37}$$

The optimization problem (1) is modified by replacing the dynamics with (36). An additional cost term,

$$\sum_{k=0}^{N-1} \frac{1}{2} w_k^T R_{\text{inf}} w_k, \tag{38}$$

and constraints $w_k = 0,\ k{=}0, \dots, N{-}1$ are also added to the problem. As the algorithm converges to feasibility, the solution approaches the same solution obtained by (1).

C. Minimum Time

Minimum time problems can be solved by considering $\tau = \sqrt{dt} \in \mathbf{R}$ as an input at each time step, $\mathrm{T} = \{\tau_0, \dots, \tau_{N-1}\}$. The optimization problem (1) is modified to use dynamics,

$$\begin{bmatrix} x_{k+1} \\ \omega_{k+1} \end{bmatrix} = \begin{bmatrix} f(x_k, u_k, \tau_k) \\ \tau_k \end{bmatrix}, \tag{39}$$

with an additional cost,

$$\sum_{k=0}^{N-1} R_{\text{min}} \tau_k^2, \tag{40}$$

and constraints $\omega_k = \tau_k,\ k{=}1, \dots, N{-}1$ to ensure time steps are equal so the solver does not exploit discretization errors in the system dynamics. Upper and lower bounds can also be placed on τ_k.

D. Projected Newton Method

The primal and dual trajectories $Y \leftarrow X, U, \lambda$ (solved to a coarse tolerance) from the first stage of ALTRO are used to warm start an active-set projected Newton method (Algorithm 5). This approach avoids the slow convergence and numerical ill-conditioning exhibited by AL methods when the penalty is made large. Typically, only one or two Newton steps are required to achieve machine-precision constraint satisfaction. To ensure strict satisfaction of the dynamics and constraints, the search direction is projected onto the constraint manifold by successively modifying the search direction, δY, of the primal variables (denoted with subscript p) at each iteration,

$$\delta Y_p \leftarrow \delta Y_p - H^T (H H^T)^{-1} h \tag{41}$$

Algorithm 5 Projected Newton

1: **function** PROJECTEDNEWTON(Y, tolerance)
2: $\bar{Y} \leftarrow$ active-set projection of Y (41)
3: $\bar{J} \leftarrow$ cost of $\bar{Y}$
4: **while** $\|\nabla \mathcal{L}\| >$ tolerance **do**
5: $Y \leftarrow \bar{Y}, J \leftarrow \bar{J}$, $\alpha = 1$
6: $\delta Y \leftarrow$ Newton search direction for Y
7: $\hat{Y} \leftarrow$ active-set projection of $Y + \alpha \delta Y$
8: $\hat{J} \leftarrow$ cost $\hat{Y}$
9: **if** $\hat{J}$ satisfies line search conditions **then**
10: $\bar{Y} \leftarrow$ multiplier projection of $\hat{Y}$ (43)
11: $\bar{J} \leftarrow$ cost of $\bar{Y}$
12: **else**
13: Reduce α, return to line 7
14: **end if**
15: **end while**
16: **return** $\bar{Y}$
17: **end function**

where h and H are the constraint violation and constraint Jacobian, respectively. This direct formulation is strictly feasible and is an anytime algorithm. Further, the dual variables (denoted with subscript d) are updated with a projection that minimizes the residual of the gradient of the Lagrangian at each itteration:

$$r = \nabla J + H^T \lambda \tag{42}$$

$$\delta Y_d = (H H^T)^{-1} H r. \tag{43}$$

Algorithm 6 ALTRO

procedure
Initialize $x_0, U,$ tolerances; $\tilde{X}$
if Infeasible Start **then**
$X \leftarrow \tilde{X}$, $W \leftarrow$ from (37)
else
$X \leftarrow$ Simulate from x_0 using U, (3),(36),(39)
end if
$(X, U), \lambda \leftarrow$ AUL A$((X, U),$ iLQR, tol.)
$(X, U) \leftarrow$ PROJECTEDNEWTON$((X, U, \lambda),$ tol.)
return X, U
end procedure

IV. Simulation Results

ALTRO's performance is compared to the classic DIRCOL method [4] on a number of benchmark motion-planning problems. Each problem uses the following cost function:

$$J = \frac{1}{2}(x_N - x_f)^T Q_f (x_N - x_f) + dt \sum_{k=0}^{N-1} \frac{1}{2}(x_k - x_f)^T Q (x_k - x_f) + \frac{1}{2} u_k^T R u_k, \tag{44}$$

is solved to constraint satisfaction $c_{\max} = 1e\text{-}4$, and is performed on a laptop computer with an Intel Core i7-6500U processor and 8GB RAM. All algorithms are implemented in

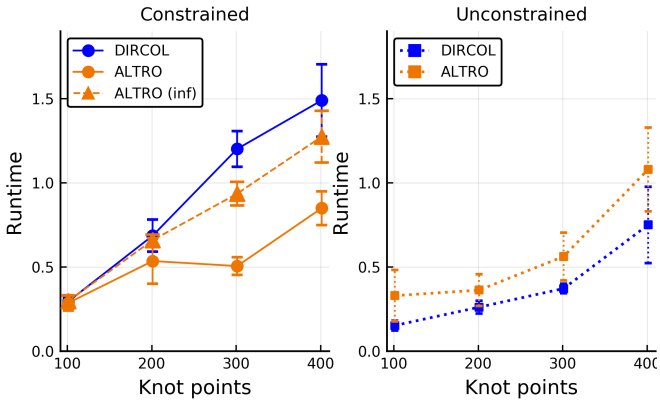

Fig. 1. Runtime comparison for parallel park problem with (left) and without (right) constraints.

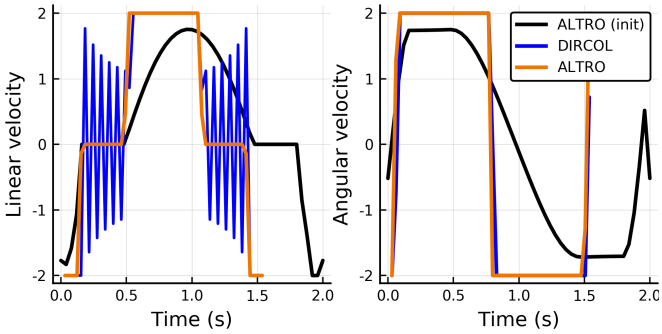

Fig. 2. Minimum Time Parallel Park. Fixed final time ($t_f = 2$s) control trajectories (black), ALTRO solution (orange), and DIRCOL solution (blue) for both linear and angular velocity control inputs. ALTRO converges to a smooth, bang-bang control, while DIRCOL exhibits rapid oscillation when linear velocity goes to zero (corresponding to the corners where the car pivots in place).

the Julia programming language. Unless otherwise specified, DIRCOL is provided the dynamically feasible state trajectory computed during the initial forward simulation from ALTRO, and solved using Ipopt [2].

A. Parallel Parking

A parallel parking problem for a Reeds-Shepp car [17] is solved with $Q = 0.001I_{3\times3}$, $R = 0.01I_{2\times2}$, $Q_f = 100I_{3\times3}$, $t_f = 3$s, and $N = 51$. Fig. 1 compares the runtime performance of ALTRO and DIRCOL. ALTRO's unconstrained runtime is typically within a standard deviation of DIRCOL. ALTRO solves faster than DIRCOL for the constrained problem. Error bars are one standard deviation and were collected using `BenchmarkTools.jl`.

A minimum-time trajectory is found by initializing both ALTRO and DIRCOL with the control trajectory of a solution with $t_f = 2$s and enforcing $-2 \leq u \leq 2$. Both algorithms converge to bang-bang control inputs and a minimum time of 1.54s (Fig. 2). Importantly, DIRCOL failed to solve the problem several times before successfully finding a solution, likely due to the way Ipopt finds an initial feasible solution. Additionally, the DIRCOL solution rapidly oscillates at turning points, unlike the ALTRO solution. While DIRCOL converged faster in this scenario, ALTRO converged more reliably (Table I).

B. Car Escape

Escape (see Fig. 3) is a problem where standard constrained iLQR fails to find an obstacle-free path to the goal. Using the same cost weights from the parallel parking problem and $N = 101$, both ALTRO and DIRCOL are initialized with a dynamically-infeasible collision-free state trajectory guessed by interpolating the six points shown in yellow in Fig. 3. ALTRO was able to converge to an optimal collision-free path in less time and fewer iterations than DIRCOL.

The projected Newton method is used to achieve high-precision constraint satisfaction after reaching a coarse threshold of maximum constraint violation $c_{\max} < 1e\text{-}3$. Fig. 4 demonstrates ALTRO* achieving $c_{\max} < 1e\text{-}8$ in one

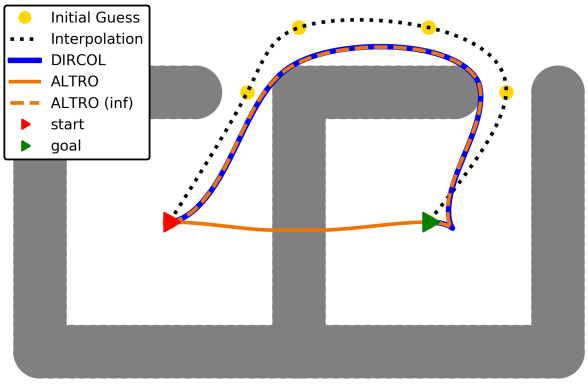

Fig. 3. Car Escape. The DIRCOL trajectory is blue, the (failed) constrained iLQR solution is solid orange, and the ALTRO solution is the dashed orange line.

Newton step. Runtime performance of this final solution-polishing step is currently relatively slow: The total runtime for ALTRO* is 1.398s (0.617s for the first stage), but should be dramatically improved with a more careful sparse matrix implementation. Further, Fig. 4 suggests that this final Newton step should be initiated once the penalty reaches its maximum value.

C. Quadrotor Maze

A quadrotor is tasked with navigating the maze (with floor and ceiling constraints) shown in Fig. 5. The cost weighting matrices are $Q = I_{13\times13}$, $R = I_{4\times4}$, $Q_f = 1000I_{13\times13}$, $t_f = 5.0$s, and $N = 101$. Input constraints, $0 \leq u \leq 50$, are also enforced. The solvers are initialized with an infeasible state trajectory indicated by the yellow points in Fig. 5 and inputs are initialized for hovering. ALTRO is able to find a collision-free trajectory, while DIRCOL fails to find collision-free trajectories even after being initialized with the solution from ALTRO (denoted by "+" in Table I).

D. Robotic Arm Obstacle Avoidance

A Kuka iiwa robotic arm is tasked with moving its end-effector through a set of closely spaced obstacles to a

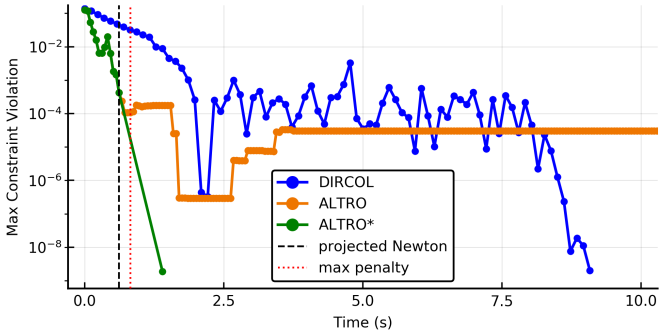

Fig. 4. Max constraint violation comparison for Escape. After reaching a coarse tolerance (dashed black line) ALTRO* performs a single iteration of Algorithm 5. DIRCOL satisfies the tolerance, but the first stage of ALTRO fails to achieve the desired constraint tolerance, likely due to poor numerical conditioning after reaching the maximum penalty (dotted red line). Additionally, DIRCOL finds a slightly lower cost than ALTRO*, 0.331 vs 0.333.

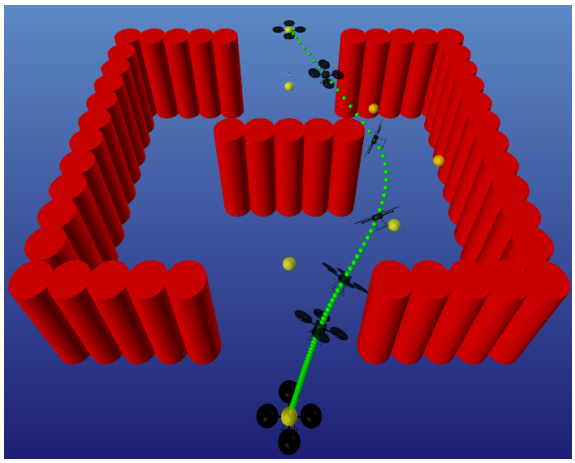

Fig. 5. Quadrotor navigating maze environment. ALTRO is initialized with a cubic interpolation of the yellow spheres and converges to the green trajectory. The quadrotor exhibits dynamic, aggressive banking maneuvers to avoid the obstacles.

desired final position (see Fig. 6) using the cost function $Q = \text{diag}(I_{7\times7}, 100I_{7\times7})$

V. Conclusion

We have presented a versatile high-performance trajectory optimization algorithm, ALTRO, that combines the advantages of both direct and indirect methods: fast convergence, infeasible state trajectory initialization, anytime dynamic feasibility, and high-precision constraint satisfaction. Our preliminary implementation demonstrates competitive—and often superior—performance on a number of benchmark motion-planning problems when compared to direct collocation implemented with the Ipopt solver. We find that ALTRO performs particularly well in comparison to DIRCOL on problems involving obstacles and high-dimensional state and input spaces. Future work will focus on improving runtime performance by taking advantage of problem sparsity and parallelization. Our implementation of ALTRO is available at `https://github.com/RoboticExplorationLab/TrajectoryOptimization.jl`.

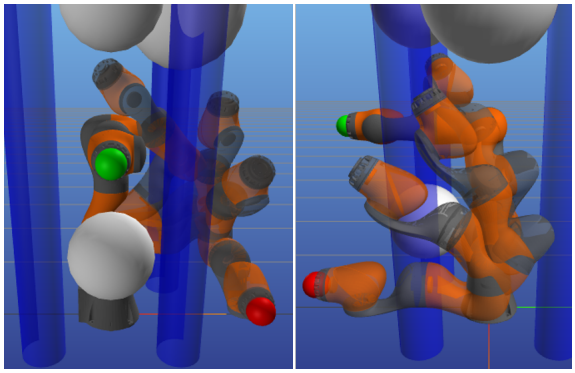

Fig. 6. Front (left) and side (right) views of Kuka iiwa arm moving end effector from initial position (red) to desired position (green).

TABLE I
Performance of ALTRO vs DIRCOL

System	Time (s)	Iters	Evals/Iter
Parallel Park	0.117 / **0.100**	42 / **24**	**410** / 532
Parallel Park	**0.098** / 0.154	**20** / 37	912 / **493**
Parallel Park (m.t.)	0.845 / **0.645**	128 / **84**	2176 / **465**
Escape	**2.83** / 9.08	**37** / 79	**357** / 948
Escape (ALTRO*)	**1.94** / 9.08	**14** / 79	**357** / 948
Quadrotor Maze	**23.0** / 559+	**218** / 5000+	**320** / 2630+
Robotic Arm	**14.3** / 313*	**227** / 4692*	1050 / **822***

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] H. B. Keller, *Numerical methods for two-point boundary-value problems*. Courier Dover Publications, 2018.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

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

[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.