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,

$$\begin{aligned} \underset{x_{0:N},\, u_{0:N-1}}{\text{minimize}} \quad & \ell_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k) \\ \text{subject to} \quad & x_{k+1} = f(x_k, u_k), \\ & x_0 = x(0), \\ & x_N = x_f, \\ & g(x_k, u_k) \leq 0, \\ & h(x_k, u_k) = 0, \end{aligned} \tag{1}$$

A straight-forward approach to solving this problem is to treat the states and inputs at each time step as decision variables and simply call a general-purpose nonlinear programming (NLP) solver such as SNOPT [1] or IPOPT [2]. Such methods are referred to as "direct," and include both direct transcription [3] (DIRTRAN) and direct collocation [4] (DIRCOL) algorithms as special cases.

Alternatively, the structure of the trajectory optimization problem can be exploited to solve a sequence of smaller subproblems using methods based on dynamic programming [5]. These algorithms typically treat only the control inputs as decision variables and enforce dynamic feasibility at each iteration by simulating the system dynamics in a "forward rollout" step. Such "indirect" methods include Differential Dynamic Programming (DDP) [5] and Iterative LQR (iLQR) [6], as well as various shooting methods [7].

In general, direct methods tend to be robust and versatile, allowing general control and state constraints. Additionally, since the states are decision variables, it is straight forward to guess an initial state trajectory, even if it is dynamically infeasible. In contrast, indirect methods solve smaller problems, making them fast and amenable to embedded systems. Due to their strict enforcement of dynamic feasibility, these methods are anytime algorithms; however, it is often difficult to find a suitable initial guess for the control trajectory, especially for high-dimensional and highly nonlinear systems. Historically, indirect methods have been considered less robust and less suitable for reasoning about general state and control constraints.

Several efforts have been made to incorporate constraints into the DDP framework. box constraints on control inputs [8] and stage inequality constraints on the states [9]–[11] 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-control constraints [12]. Augmented Lagrangian (AL) methods have been proposed [13], including hybrid approaches that also solve constrained QPs for stage state-input constraints [10], [14]. Additionally, continuous-time state-input constraints have been handled by solving a KKT system while using a penalty method for state constraints [15].

In this paper we present ALTRO (Augmented Lagrangian TRajectory Optimizer), a trajectory optimization algorithm that combines the best characteristics of both methods, namely: speed, small problem size, numerical robustness, handling of general state and input constraints, anytime dynamic feasibility, and infeasible state trajectory initialization capability. Using iLQR in an AL framework to handle general state and input constraints, we: 1) derive a numerically robust square-root formulation of the BP, 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 result is a state-of-the-art trajectory optimization algorithm that combines the advantages of both direct and indirect methods.

The paper is organized as follows: Section II provides

[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

background for iLQR and AL methods. Section III derives the constrained trajectory optimization algorithm ALTRO. Comparisons between ALTRO and DIRCOL are performed for several simulated systems in Section IV. Finally, we summarize our findings in Section V.

II. Background

Notation: For a function $f(x,u)$ we define $f_x \equiv \partial f(x)/\partial x$, $f_{xx} \equiv \partial^2 f(x,u)/\partial x^2$, and $f_{xu} \equiv \partial^2 f(x,u)/\partial x \partial u$. 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,

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

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, $u \in \mathbf{R}^m$ is a control input, and dt is the time step. The dynamics constraints are handled implicitly by using an initial state x_0 and nominal control trajectory, $U = \{u_0, \dots, u_{N-1}\}$, to simulate the state trajectory $X = \{x_0 \dots, x_N\}$.

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

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

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

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

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

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

resulting in the optimal terminal cost-to-go,

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

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

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

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

where the block matrices are,

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

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

$$Q_{ux} = \ell_{ux} dt + f_u^T P_{k+1} f_x \tag{13}$$

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

$$Q_u = \ell_u dt + f_u^T p_{k+1}. \tag{15}$$

Minimizing (10) with respect to δu gives the correction to the control trajectory. The result is a feedforward term d and a linear feedback term $K\delta x$. Regularization is added to ensure the invertibility of Q_{uu},

$$\delta u^* = -(Q_{uu} + \rho I)^{-1}(Q_{ux}\delta x + Q_u) \equiv K\delta x + d \tag{16}$$

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

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

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

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

A Forward Pass (FP), see Algorithm 3, simulates the system using the correction to the nominal control trajectory. A line search is performed on the feedforward term d to ensure a reduction in cost.

Algorithm 1 Iterative LQR

1: Initialize x_0, U, ϵ
2: $X \leftarrow$ Simulate from x_0 using U, (3)
3: **function** iLQR(X, U)
4: $J \leftarrow$ Using X, U, (2), $J^- \leftarrow \infty$
5: **while** $|J - J^-| > \epsilon$ **do**
6: $J^- \leftarrow J$
7: $K, d, \Delta V \leftarrow$ BackwardPass(X, U)
8: $X, U, J \leftarrow$ ForwardPass($X, U, K, d, \Delta V, J^-$)
9: **end while**
10: **return** X, U, J
11: **end function**

Algorithm 2 Backward Pass

1: **function** BackwardPass(X, U)
2: $p_N, P_N \leftarrow (8), (9)$
3: **for** k=N-1:-1:0 **do**
4: $\delta Q \leftarrow (10)$
5: **if** $Q_{uu} \succ 0$ **then**
6: $K, d, \Delta V \leftarrow (16), (19)$
7: **else**
8: Increase ρ and go to line 3
9: **end if**
10: **end for**
11: **return** $K, d, \Delta V$
12: **end function**

B. Augmented Lagrangian

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

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

where $\mathcal{I}$ and $\mathcal{E}$ are the index sets corresponding to the inequality and equality constraints, respectively. AL methods

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**
    $\bar{u}_k = u_k + K(\bar{x}_k - x_k) + \alpha d_k$
    $\bar{x}_{k+1} \leftarrow$ 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, Φ, ϵ)
  Initialize λ, μ
  while $\max(c) > \epsilon$ **do**
    Minimize $\mathcal{L}_A(x, \lambda, \mu)$ w.r.t. x using Φ
    Update λ using (23), update μ using (24)
  end while
return X, λ
end function

solve (20) by converting it to an unconstrained optimization problem and adding a penalty function to the Lagrangian,

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

where I_μ is a diagonal matrix defined as,

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

An unconstrained optimization method is used to solve the AL to (approximate) convergence, holding λ and μ constant. Taking the gradient of (21) with respect to the primal variable suggests the following dual update,

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

while the penalty is increased monotonically according to some 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. Theoretically, convergence of the AL is superlinear while the penalty term is increased to a large value. In practice, the convergence rate becomes linear after the penalty is capped at this maximum finite value [16]. DDP has previously been used to solve the inner minimization of an AL with good results [10], [13].

III. ALTRO Algorithm

ALTRO, see Algorithm 6, comprises two stages: the primary stage solves (1) rapidly to a coarse tolerance (e.g., $\sqrt{\epsilon}$) using iLQR to solve the unconstrained sub-problems within an AL framework, similar to [13]. The optional secondary stage uses the primal and dual estimates from the first stage to warm start an active-set method that achieves machine-precision constraint tolerance. We present several refinements and extensions to previous work on constrained iLQR, as well as the novel secondary "solution polishing" stage.

A. Square-Root Backward Pass

For AL methods to achieve superlinear convergence the penalty terms must be increased to large values. Even for finite penalty values, this can result in large ill-conditioning. To help mitigate this issue, we introduce a numerically robust BP inspired by the square-root Kalman filter, a standard tool for state estimation.

We derive expressions for the following matrices: $S \equiv \sqrt{P}, Z_{xx} \equiv \sqrt{Q_{xx}}$, and $Z_{uu} \equiv \sqrt{Q_{uu}}$. For compactness, we drop the dependence on time step k, and denote P_{k+1} as P'. The terminal cost-to-go Hessian is simply $S_N = \sqrt{P_N}$ and the action-value expansion factorizations are,

$$Z_{xx} \leftarrow \text{QR}\left(\begin{bmatrix} \sqrt{\ell_{xx}}, S' f_x, \sqrt{I_\mu} c_x \end{bmatrix} \right) \tag{25}$$

$$Z_{uu} \leftarrow \text{QR}\left(\begin{bmatrix} \sqrt{\ell_{uu}}, S' f_u, \sqrt{I_\mu} c_u, \sqrt{\rho} I \end{bmatrix} \right), \tag{26}$$

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

$$K = -Z_{uu}^{-1} Z_{uu}^{-T} Q_{ux} \tag{27}$$

$$d = -Z_{uu}^{-1} Z_{uu}^{-T} Q_u, \tag{28}$$

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

$$p = Q_x + (Z_{uu} K)^T (Z_{uu} d) + K^T Q_u + Q_{xu} d \tag{29}$$

$$\Delta V = d^T Q_u + \frac{1}{2} (Z_{uu} d)^T (Z_{uu} d). \tag{30}$$

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

$$P = \begin{bmatrix} I \\ K \end{bmatrix}^T \begin{bmatrix} Z_{xx}^T & 0 \\ C^T & D^T \end{bmatrix} \begin{bmatrix} Z_{xx} & C \\ 0 & D \end{bmatrix} \begin{bmatrix} I \\ K \end{bmatrix}$$

$$= \begin{bmatrix} Zxx + CK \\ DK \end{bmatrix}^T \begin{bmatrix} Z_{xx} + CK \\ DK \end{bmatrix}$$

$$S \leftarrow \text{QR}\left(\begin{bmatrix} Z_{xx} + CK \\ DK \end{bmatrix} \right), \tag{31}$$

where,

$$C = Zxx^{-T} Q_{ux} \tag{32}$$

$$D = \sqrt{Z_{uu}^T Z_{uu} - C^T C}. \tag{33}$$

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 $w \in \mathbf{R}^n$,

$$x_{k+1} = f(x_k, u_k) + w_k, \tag{34}$$

to make the system artificially fully actuated.

Given 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:

$$w_k = \tilde{x}_{k+1} - f(x_k, u_k) \tag{35}$$

The optimization problem (1) is modified to have dynamics (34), an additional cost term,

$$\sum_{k=0}^{N-1} \frac{1}{2} w_k^T R_{\text{inf}} w_k, \tag{36}$$

and constraints $w_k = 0,\ k{=}0, \dots, N{-}1$. 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 $\tau = \sqrt{dt}$ as an input at each time step, $\mathrm{T} = \{\tau_0, \dots, \tau_{N-1}\}$. The optimization problem (1) is modified to use dynamics,

$$\begin{bmatrix} x_{k+1} \\ \omega_{k+1} \end{bmatrix} = \begin{bmatrix} f(x_k, u_k, \tau_k) \\ \tau_k \end{bmatrix}, \tag{37}$$

with an additional cost,

$$\sum_{k=0}^{N-1} R_{\text{min}} \tau_k^2, \tag{38}$$

constraints $\omega_k = \tau_k,\ k{=}1, \dots, N{-}1$ to ensure time steps are equal so the solver does not exploit the system dynamics, and upper and lower bounds on τ_k.

D. Projected Newton Method

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

$$\delta Y_p \leftarrow \delta Y_p - H^T (H H^T)^{-1} h \tag{39}$$

at each iteration, 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,

$$r = \nabla J + H^T \lambda \tag{40}$$

$$\delta Y_d = (H H^T)^{-1} H r. \tag{41}$$

Algorithm 5 Projected Newton

```
function ProjectedNewton(Y, ϵ)
    Ȳ ← active-set projection of Y (39)
    J̄ ← cost of Ȳ
    while ‖∇ℒ‖ > ϵ do
        Y ← Ȳ, J ← J̄, α = 1
        δY ← Newton search direction for Y
        Ŷ ← active-set projection of Y + αδY
        Ĵ ← cost Ŷ
        if Ĵ satisfies line search conditions then
            Ȳ ← multiplier projection of Ŷ (41)
            J̄ ← cost of Ȳ
        else
            Reduce α, return to line 7
        end if
    end while
return Ȳ
end function
```

Algorithm 6 ALTRO

```
procedure
    Initialize x_0, U, λ, μ, ρ, ϕ, ϵ_J, ϵ_c, ϵ_N; X̃
    if Infeasible Start then
        X ← X̃, W ← from (35)
    else
        X ← Simulate from x_0 using U
    end if
    (X, U), λ ← AuLa((X, U), iLQR, ϵ_c)
    (X, U) ← ProjectedNewton((X, U, λ), ϵ_N)
return X, U
end procedure
```

IV. Simulation Results

Each problem uses the following cost function:

$$J = \frac{1}{2}(x_N - x_f)^T Q_f (x_N - x_f) + dt \sum_{k=0}^{N-1} \frac{1}{2}(x_k - x_f)^T Q (x_k - x_f) + \frac{1}{2} u_k^T R u_k, \tag{42}$$

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

A. Dubins Car Parallel Park

A parallel park problem for a Dubins car is solved with $Q = \text{diag}_{3\times3}(0.001)$, $R = \text{diag}_{2\times2}(0.01)$, $Q_f = \text{diag}_{3\times3}(100)$, $t_f = 3$s, and $N = 51$. Fig. 2 shows the control trajectories and Fig. 1 compares the runtime performance

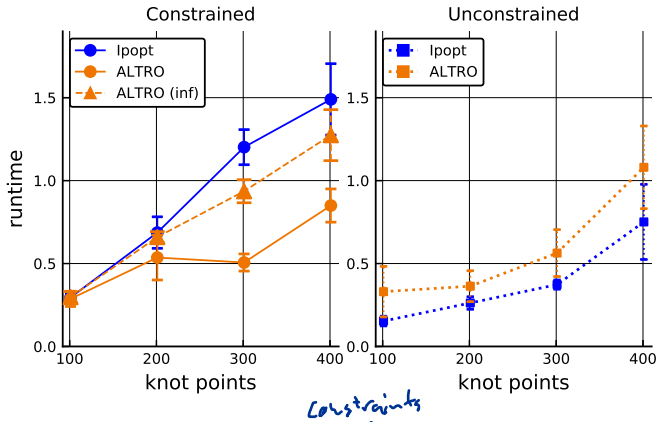

Fig. 1. Runtime comparison for parallel park problem with constrained (left) and unconstrained (right) solves. ALTRO's unconstrained runtime is typically within a standard deviation of DIRCOL, while ALTRO's constrained runtime performs better than DIRCOL. However, ALTRO with infeasible start (which solves a larger problem) is relatively slower. Error bars are one standard deviation and were collected using `BenchmarkTools.jl`.

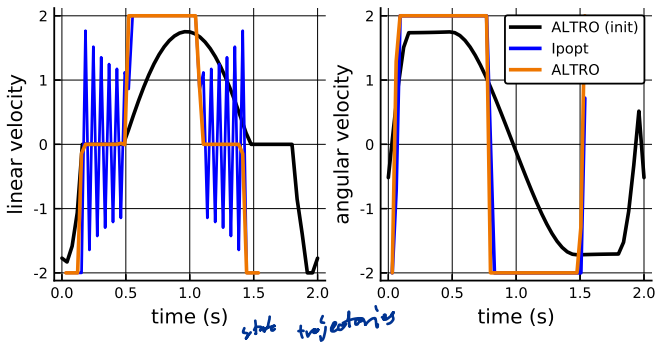

Fig. 2. Minimum Time Parallel Park. Fixed final time ($t_f = 2$s) control trajectories (black), ALTRO solution (orange), and DIRCOL solution (blue) for both linear and angular velocity commands. ALTRO converges to a smooth, bang-bang control, while DIRCOL exhibits rapid oscillation when linear velocity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

of ALTRO and DIRCOL. 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 $-2 \preceq u \preceq 2$. Both algorithms converge to bang-bang control and a minimum time of 1.55 and 1.54 seconds, respectively. However, DIRCOL appears to suffer from numerical issues when the linear velocity goes to zero. Additionally, DIRCOL also failed to solve the problem several times before successfully finding a solution, likely due to the way Ipopt finds an initial feasible solution. While DIRCOL converged faster than ALTRO in this scenario, the trade-off is inconsistency and quality of the solution. See Table I for more details.

B. Dubins 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 Q, R, Q_f, t_f from the parallel park problem and $N = 101$, both ALTRO and DIRCOL are initialized with a cubic interpolated state trajectory from 6 knot points and find collision free trajectories. ALTRO converges in less

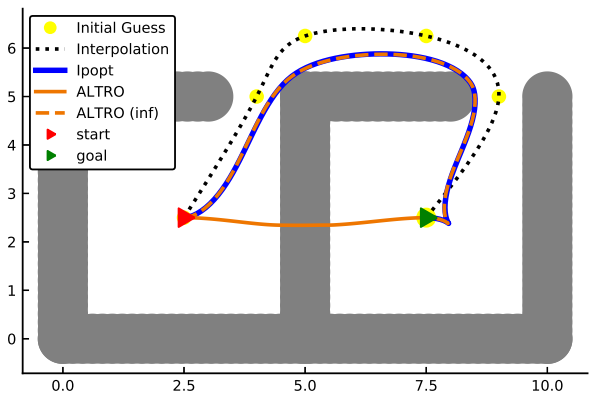

Fig. 3. Dubins Car Escape. The DIRCOL (Ipopt) trajectory is blue, the (failed) constrained iLQR solution is solid orange, and the ALTRO solution is the dashed orange line.

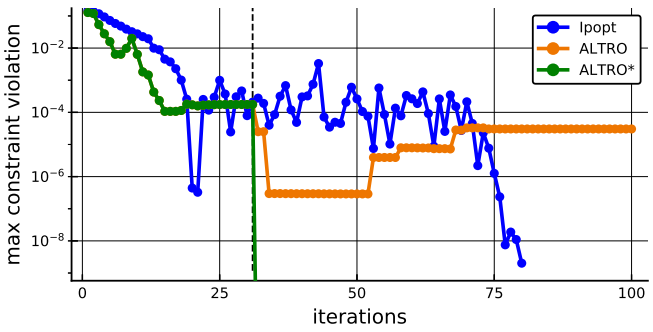

Fig. 4. Max constraint violation comparison for Escape. After reaching a coarse tolerance, $c_{\max} < 1e\text{-}4$, ALTRO* performs a single iteration of Algorithm 5 and achieves $c_{max} \ll 1e\text{-}8$. Ipopt satisfies the tolerance, but the primary stage of ALTRO fails to achieve the desired constraint satisfaction, likely due to poor numerical conditioning. Additionally, Ipopt finds a slightly lower cost than ALTRO*, 0.331 vs 0.332.

time and iterations.

The projected Newton method is used to achieve machine-precision constraint satisfaction for Escape after reaching a coarse threshold of $1e\text{-}4$. Figure **??** demonstrates the the runtime performance where ALTRO converges the in one step. Runtime performance of the iterate is relatively slow, but should be dramatically improved with future, more careful, implementations.

C. Quadrotor Maze

A quadrotor with thirteen state dimensions (quaternion angular representation) and four controls is tasked with navigating a maze (with floor and ceiling constraints), see Fig. 5. The cost function is $Q = \text{diag}_{13\times13}(1)$, $R = \text{diag}_{4\times4}(1)$, $Q_f = \text{diag}_{13\times13}(1000)$, $t_f = 5.0$s, and $N = 101$, subject to control constraints: $0 \preceq u \preceq 50$. Initialized with a cubic interpolated state trajectory with 7 knot points, and controls for hovering, ALTRO is able to find a collision-free trajectory. Constrained iLQR fails to find a collision free trajectory, and interestingly, DIRCOL fails to converge to a solution.

D. Robotic Arm Obstacle Avoidance

A Kuka iiwa robotic arm is tasked with moving its end-effector through tight obstacles to a desired position (see

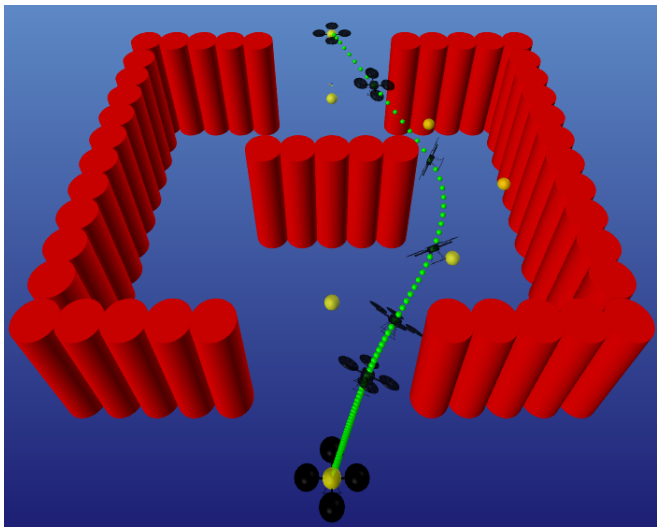

Fig. 5. Quadrotor navigating maze environment. ALTRO is initialized with a cubic interpolation of the yellow spheres and converges to the green trajectory. The quadrotor exhibits dynamic, aggressive banking maneuvers to avoid the obstacles.

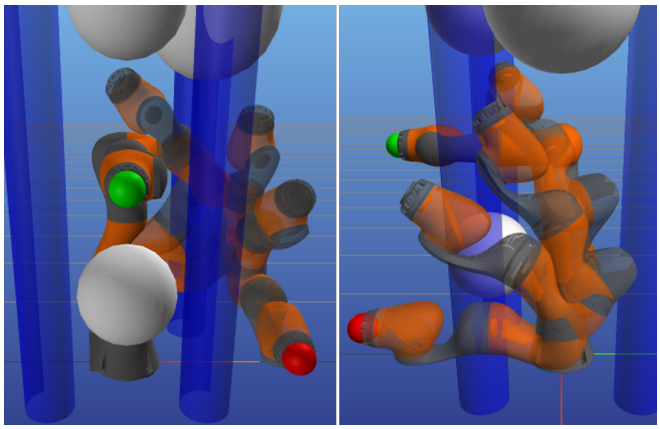

Fig. 6. Front (left) and side (right) views of Kuka iiwa arm moving end effector from initial position (red) to desired position (green).

Fig. 6) using the cost function $Q = \text{diag}_{14\times14}(1_7, 100_7)$, $R = \text{diag}_{7\times7}(0.01)$, $Q_f = \text{diag}_{14\times14}(10)$, $t_f = 5.0$s, and $N = 41$, subject to torque limits: $-80 \preceq u \preceq 80$. The control trajectory is initialized with a ~~holding trajectory.~~ Ipopt failed to solve this problem unless initialized with the ALTRO solution. Performance of ALTRO and warm-started Ipopt (values denoted *) are given in Table I.

V. Conclusion

We presented a versatile, ~~hybrid~~ trajectory optimization algorithm, ALTRO, that leverages the advantages of both direct and indirect methods: speed, general constraints, state trajectory initialization, anytime dynamic feasibility, and machine-precision constraint satisfaction. Our preliminary implementation demonstrates competitive—and often superior—performance on a number of benchmark robotics problems compared to a conventional direct collocation method using Ipopt. We find that ALTRO is particularly good at problems involving obstacles and systems with many states and controls, where Ipopt generally performs poorly. Future work will improve runtime performance with a more careful implementation and parallelization, ~~and perform a more comprehensive comparison–including against a commerical SQP solver such as SNOPT. This~~ our implementation, written in Julia, is available at `https://github.com/RoboticExplorationLab/TrajectoryOptimization.jl`.

TABLE I

Performance of ALTRO vs DIRCOL

System	Time (s)	Iters	Evals/Iter
Parallel Park	xx / xx	42 / **24**	**410** / 532
Parallel Park (Con.)	xx / xx	**20** / 37	912 / **493**
Parallel Park (M.T.)	xx / xx	266 / **84**	2176 / **465**
Escape	xx / xx	**37** / 79	**357** / 948
Quadrotor Maze	**xx** / 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] D. M. Murray and S. J. Yakowitz, "Constrained differential dynamic programming and its application to multireservoir control," en, *Water Resources Research*, vol. 15, no. 5, pp. 1017–1027, Oct. 1979.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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