Control-Minimal Time-Assigned Path-Constrained Trajectory Optimization

Adam Morrissett and Patrick J. Martin

***Abstract*—Path-constrained trajectory optimization research normally focuses on time or energy optimality. However, some applications seek reference trajectories that satisfy other constraints. In this paper, we formulate a control-minimal time-assigned path-constrained trajectory optimization problem: a mobile ground robot must traverse a given path in a specific amount of time using minimal control effort. Through a nonlinear change of variables, we solve this problem using convex optimization. We evaluate our solution with an intelligent transportation scenario where an autonomous vehicle must cross an intersection in a specific amount of time while following the turn lane's geometric center.**

I. Introduction

Road networks are generally well-defined structures where autonomous vehicles need to follow two high-level rules: 1) avoid collisions; and 2) traverse the path (trip route) in minimal time, with minimal control, or a combination of these goals. Some transportation applications benefit from predictable, but not necessarily minimal, timing. In particular, scheduled transportation tasks with specific timing requirements may be the primary objective. Consider a bus route comprising several stops, each with a pre-arranged departure time. If the bus arrives too early, it wastes time and energy waiting at its stop. If it arrives too late, it will cause disruptions in the schedule. In such a use case, the bus's reference trajectory and control signals should ensure the bus arrives as close to the specified arrival time as possible.

One can generalize this scheduling concept by considering intersection management systems (IMS) that rely on accurate timing to predict vehicles' traversal times and safely schedule crossings [1]. In previous work, we developed an auction-based IMS in which vehicles bid using their cost functions [2]. The vehicles' motion planners generated reference trajectories and control inputs to drive through the intersection in so-called *crossing times*. They also calculated a cost for each crossing time and used those costs to define a *crossing cost* function. The management system used these functions to assign final crossing times to each vehicle.

This material is based upon work supported by the Federal Highway Administration under Agreement No. 693JJ32245201. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the Author(s) and do not necessarily reflect the view of the Federal Highway Administration.

The authors are with the Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, Virginia 23284-3068 (email: morrissettal2@vcu.edu; martinp@vcu.edu).

For each candidate crossing time, vehicles solved a trajectory optimization problem. Their objective functions comprised a crossing time cost, a reference tracking cost, and a control effort cost. This formulation sufficed for longer candidate crossing times, but it caused issues when evaluating shorter ones. Vehicles' motion plans significantly deviated from their reference paths in order to satisfy the crossing time constraint, sacrificing path tracking for a lower control cost. These reference tracking deviations often made the vehicles move into dangerous situations, such as cutting over sidewalks. To ensure that the IMS schedule assignments would truly be safe, we need vehicles to generate control inputs that kept them on their reference paths while also satisfying the IMS time assignment.

The above challenge motivated the work presented in this paper. The motion planners still seek an effort-minimal reference trajectory to traverse the path. However, we reformulate the path tracking goal to a hard constraint so that the planners consider only control inputs that keep the vehicle on the path. We kept the crossing time constraint, meaning vehicles have a finite time to traverse their path. Using this new approach, *control-minimal time-assigned path-constrained trajectory optimization*, the IMS may more accurately compare crossing costs.

Solution categories for path-constrained trajectory planning include dynamic programming [3, Sec. 14.6.3], numerical integration [3, Sec. 14.6.3], reachability analysis (RA) [4], and convex optimization [5]. Numerical integration and convex optimization methods are particularly useful when performing time- or energy-optimal planning for robotic arms [5], biped robots [6], and mobile ground robots [7], [8].

Time-optimal trajectory planning [9]–[11] tries to minimize the path traversal time, while energy-optimal planning typically disregards it. In contrast to those goals, we seek the lowest control effort needed to move a robot along a path in a specific duration. Compared to other problem formulations, the time-assigned variant has received minimal attention.

Researchers in [12] were one of the first to study time-assigned path-constrained trajectory planning, and they solved the problem using nonlinear semi-infinite programming. More recently, the authors of [4] extended their RA framework for robotic arms to find trajectories of specified duration.[1] Their algorithm searches for a constraint-satisfying

[1] https://web.archive.org/web/20220421233339/https://hungpham2511.github.io/toppra/python_api.html

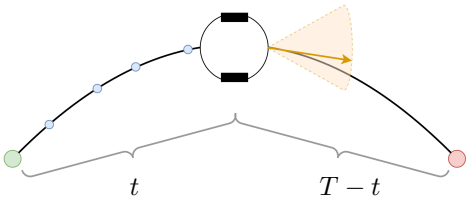

Fig. 1. The robot traverses the path (black) starting from the beginning (green) and stopping at the end (red). Blue points are the robot's planning history. The planner chooses control inputs (orange arrow) to keep the robot on the path. The orange cone represents possible control inputs. The robot traverses the path in T seconds.

deformation between the time-maximal and time-minimal trajectories. However, the final result may not be control-minimal.

Another work developed a speed planner for a heavy vehicle crossing an intersection [11]. To avoid collisions, the optimization problem imposed time window constraints. The vehicle's crossing duration had to be within one of these windows. Our proposed formulation, in contrast, requires the vehicle to cross in a specific amount of time.

Compared to the other solution methods, convex optimization provides a framework that allows for flexible customization of objectives and constraints. Additionally, several solvers and front-ends exist that that ease its implementation. In this paper, we use a nonlinear change of variables to reformulate the original, nonlinear trajectory optimization problem as a convex one, as done in [5] for time-optimal planning for robotic arms. We then use a collocation method to approximate the solution by converting the problem into a second-order cone program (SOCP). We evaluate our solution on several curves that vehicles commonly encounter while driving, particularly at intersections. This paper's contributions are 1) an application of path-constrained trajectory optimization to mobile ground robots; 2) a formulation of a second-order unicycle motion model; and 3) an SOCP solution to the time-assigned path-constrained trajectory optimization problem variant. Note that while this paper focuses on autonomous vehicles, the methods we present generalize to various mobile ground robots.

II. Problem Formulation

Consider the problem visualized in Fig. 1. We want a mobile ground robot to traverse a path close to a specific time while minimizing its control effort. The robot must start from the path's beginning, stop at its end, and stay on this path while moving.

A. Robot Motion Model

We model the robot in Fig. 1 as a rigid body moving on a plane, making its configuration space $\mathcal{C}$ equal to the special Euclidean group $SE(2)$. The robot's configuration $\boldsymbol{q} \in \mathcal{C}$ is $\boldsymbol{q} \coloneqq (x, y, \theta)$, where x and y represent the robot's position on the plane, and θ is its heading with respect to the x-axis.

The robot moves according to a unicycle motion model under no-slip conditions. The configuration and its components are implicit functions of time $t \in \mathbb{R}_+$, where $\mathbb{R}_+$ is the set of nonnegative real numbers. The following system of equations defines the configuration transformation for a single-order kinematic unicycle [3, eq. (13.18)]:

$$\dot{x} = v\cos\theta \qquad \dot{y} = v\sin\theta \qquad \dot{\theta} = \omega \tag{1}$$

where v is the robot's scalar linear velocity (forward is positive), and ω is its scalar angular velocity (counterclockwise is positive). Mark $\dot{\square}$ denotes the first time-derivative.

We could use the linear and angular scalar velocities of (1) as control inputs, but the resulting system would be unrealistic as robots do not start and stop instantaneously. Differentiating (1) with respect to time results in the second-order kinematic unicycle. The following system of equations defines the new configuration transformation:

$$\begin{aligned} \ddot{x} &= u_{t,\text{lin}}\cos\theta - \dot{\theta}\, v_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\sin\theta \\ \ddot{y} &= u_{t,\text{lin}}\sin\theta + \dot{\theta}\, v_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\cos\theta \\ \ddot{\theta} &= u_{t,\text{ang}}, \end{aligned} \tag{2}$$

where $u_{t,\text{lin}}$ is the scalar linear acceleration, $u_{t,\text{ang}}$ is the scalar angular acceleration, and $\ddot{\square}$ denotes the second time-derivative. We choose the scalar linear and angular accelerations as the system's inputs $\boldsymbol{u}_t \coloneqq (u_{t,\text{lin}}, u_{t,\text{ang}})$. Subscript $\square_t$ indicates that the system controls vary with time. Function $v_{\boldsymbol{q}} \colon \mathcal{C} \times \mathbb{R}^3 \to \mathbb{R}$ is the robot's scalar linear velocity in terms of its configuration and first time-derivative $\dot{\boldsymbol{q}}$. It is given by

$$v_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \sqrt{\dot{x}^2 + \dot{y}^2}.$$

We use the $\square_{\boldsymbol{q}}$ subscript to denote functions of the robot's configuration and its derivatives.

Equation (2) is control-affine and may be restructured as

$$\ddot{\boldsymbol{q}} = \boldsymbol{f}_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{G}_{\boldsymbol{q}}(\boldsymbol{q})\boldsymbol{u}_t, \tag{3}$$

where

$$\boldsymbol{f}_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{bmatrix} -\dot{\theta} v_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\sin\theta \\ \dot{\theta} v_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\cos\theta \\ 0 \end{bmatrix} \qquad \boldsymbol{G}_{\boldsymbol{q}}(\boldsymbol{q}) = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix}$$

B. Path-Constrained Motion

We define a path as a continuous, collision-free function $\boldsymbol{\tau} \colon [0, 1] \to \mathcal{C}_{\text{free}}$ that maps a point $s \in [0, 1]$ to a robot configuration. Set $\mathcal{C}_{\text{free}}$ is the subset of the robot's configuration space that excludes obstacles. Point s, which we term the *path-position*, is the robot's position along the path. Value $\boldsymbol{\tau}(s = 0)$ represents the path's beginning, and $\boldsymbol{\tau}(s = 1)$ is its end. The path-position is an implicit function of time.

Suppose we constrain the robot's motion to a dynamically-feasible path. We express its configuration in terms of its path-position [3, Sec. 14.6.3]:

$$\boldsymbol{q} = \boldsymbol{\tau}(s). \tag{4}$$

We also relate the robot's velocity $\dot{\boldsymbol{q}}$ to the path by differentiating (4) with respect to time:

$$\dot{\boldsymbol{q}} = \frac{\mathrm{d}\boldsymbol{\tau}}{\mathrm{d}s}\frac{\mathrm{d}s}{\mathrm{d}t} = \boldsymbol{\tau}'(s)\dot{s}, \tag{5}$$

where $\square'$ denotes the first derivative with respect to the path-position, and $\dot{s}$ is the robot's *path-velocity*.

Furthermore, we express the robot's path-constrained acceleration by differentiating (5) with respect to time:

$$\ddot{\boldsymbol{q}} = \frac{\mathrm{d}^2\boldsymbol{\tau}}{\mathrm{d}s^2}\dot{s}^2 + \frac{\mathrm{d}\boldsymbol{\tau}}{\mathrm{d}s}\ddot{s} = \boldsymbol{\tau}''(s)\dot{s}^2 + \boldsymbol{\tau}'(s)\ddot{s}, \tag{6}$$

where $\square''$ denotes the second derivative with respect to the path-position, and $\ddot{s}$ is the robot's *path-acceleration*.

Now that we have the robot's configuration expressed in terms of the path, we substitute (4)–(5) into (3) to derive the path-constrained system dynamics:

$$\begin{aligned}\ddot{\boldsymbol{q}} &= \boldsymbol{f}_{\boldsymbol{q}}\big(\boldsymbol{\tau}(s), \boldsymbol{\tau}'(s)\dot{s}\big) + \boldsymbol{G}_{\boldsymbol{q}}\big(\boldsymbol{\tau}(s)\big)\boldsymbol{u}_t \\ &= \boldsymbol{f}_s(s,\dot{s}) + \boldsymbol{G}_s(s)\boldsymbol{u}_t,\end{aligned} \tag{7}$$

where

$$\boldsymbol{f}_s(s,\dot{s}) = \begin{bmatrix} -\tau_3'(s)\,\dot{s}\,v_s(s,\dot{s})\,\sin(\tau_3(s)) \\ \tau_3'(s)\dot{s}v_s(s,\dot{s})\cos(\tau_3(s)) \\ 0 \end{bmatrix},$$

$$\boldsymbol{G}_s(s) = \begin{bmatrix} \cos(\tau_3(s)) & 0 \\ \sin(\tau_3(s)) & 0 \\ 0 & 1 \end{bmatrix},$$

and

$$v_s(s,\dot{s}) = \sqrt{\left[\tau_1'(s)\dot{s}\right]^2 + \left[\tau_2'(s)\dot{s}\right]^2}. \tag{8}$$

Subscript $\square_s$ denotes functions of the robot's path-position and its derivatives. Notation $\tau_i^{\square}$ denotes the i^{th} component of the path or its derivative. Furthermore, we equate (6) and (7) to express the path-constrained dynamics entirely in terms of the path-position, its derivatives, and the system input:

$$\boldsymbol{\tau}''(s)\dot{s}^2 + \boldsymbol{\tau}'(s)\,\ddot{s} = \boldsymbol{f}_s(s,\dot{s}) + \boldsymbol{G}_s(s)\boldsymbol{u}_t. \tag{9}$$

We now define the basic control-minimal, time-assigned, path-constrained trajectory optimization problem:

$$\begin{aligned}
\min_{s,\,\boldsymbol{u}_t} \quad & \int_0^T \|\boldsymbol{u}_t(t)\|_2^2\,\mathrm{d}t && \text{(10a)}\\
\text{s.t.} \quad & \boldsymbol{\tau}''(s)\dot{s}^2 + \boldsymbol{\tau}'(s)\ddot{s} = \boldsymbol{f}_s(s,\dot{s}) + \boldsymbol{G}_s(s)\boldsymbol{u}_t, && \text{(10b)}\\
& \underline{\boldsymbol{u}} \le \boldsymbol{u}_t \le \overline{\boldsymbol{u}}, && \text{(10c)}\\
& \text{for } t \in [0,T], \\
& s(0) = 0, && \text{(10d)}\\
& s(T) = 1 && \text{(10e)}
\end{aligned}$$

where $\underline{\boldsymbol{u}}$ and $\overline{\boldsymbol{u}}$ are the lower and upper actuator limits for $\boldsymbol{u}_t$, respectively. Constraint (10b) restricts the robot's motion to the path, (10c) bounds the control inputs, and (10d)–(10e) define the boundary constraints. The traversal time T is a design parameter that specifies how long the robot has to traverse the path.

We want to find an optimal path-position function $s^\star$, which defines a time scaling along the path [3, Sec. 7.1.3], and an optimal control input function $\boldsymbol{u}_t^\star$ that achieves the time scaling. In contrast to time-minimal or energy-minimal problem variants, we seek a time scaling that causes the robot to finish traversing the path in exactly T seconds.

The resulting control signal could be fed into the system. Alternatively, the time scaling could be composed with the path to form a reference trajectory that is passed into a tracking controller.

C. Convex Reformulation

Optimization problem (10) is nonlinear in the path-position and its derivatives, but we reformulate it into a convex problem using a nonlinear change of variables. We follow the reformulation in [5], which presented a convex optimization solution for time-minimal, path-constrained trajectory optimization with a robotic arm. Our work studies the time-assigned problem variant for a mobile robot.

Before introducing the nonlinear change of variables, we simplify the path-constrained dynamics from (9). Function v_s is linear in the path-velocity, so we extract $\dot{s}$ from the radical in (8) to form (with a slight abuse of notation) an alternate equation:

$$v_s(s,\dot{s}) = \dot{s}v_s(s),$$

where

$$v_s(s) = \sqrt{\left[\tau_1'(s)\right]^2 + \left[\tau_2'(s)\right]^2}. \tag{11}$$

By substituting (11) into (9), we express the path-constrained system dynamics with a more concise model:

$$\boldsymbol{h}_s(s)\ddot{s} = \boldsymbol{f}_s(s)\dot{s}^2 + \boldsymbol{G}_s(s)\boldsymbol{u}_t,$$

where

$$\boldsymbol{h}_s(s) = \boldsymbol{\tau}'(s),$$

and

$$\boldsymbol{f}_s(s) = \begin{bmatrix} -\tau_1''(s) - \tau_3'(s)v_s(s)\sin(\tau_3(s)) \\ -\tau_2''(s) + \tau_3'(s)v_s(s)\cos(\tau_3(s)) \\ -\tau_3''(s) \end{bmatrix}.$$

Next, we change the objective function's integration variable from time (t) to the path-position (s):

$$\begin{aligned}\int_0^T \|\boldsymbol{u}_t(t)\|_2^2\,\mathrm{d}t &= \int_{s(0)}^{s(T)} \frac{\|\boldsymbol{u}_t(t)\|_2^2}{\mathrm{d}s/\mathrm{d}t}\,\mathrm{d}s \\ &= \int_0^1 \frac{\|\boldsymbol{u}_s(s)\|_2^2}{\dot{s}}\,\mathrm{d}s\,.\end{aligned}$$

As a consequence of changing the integration variable, we must also introduce a new control input $\boldsymbol{u}_s\colon [0,1] \to \mathbb{R}^2$ that is a function of the path-position instead of time. Propagating the change of variables into the rest of the optimization problem, we treat s, $\dot{s}$, and $\ddot{s}$ as decision variables.

Finally, we introduce the nonlinear change of variables [5, eqs. (17) and (18)]:

$$z(s) \coloneqq \dot{s}^2 \qquad \nu(s) \coloneqq \ddot{s}.$$

In the resulting differential-algebraic system of equations (DAE), z is the differential state, $\boldsymbol{u}_s$ is the algebraic state, and ν is the control input. As derived in [5], the system has linear dynamics defined by [5, eq. (19)]

$$z'(s) = 2\nu(s).$$

With the change of variables, we redefine problem (10) as a convex one:

$$\begin{aligned}
\underset{z,\boldsymbol{u}_s,\nu}{\text{min.}} \quad & \int_0^1 \frac{\|\boldsymbol{u}_s(s)\|_2^2}{\sqrt{z(s)}}\,\mathrm{d}s && \text{(12a)}\\
\text{s.t.} \quad & \boldsymbol{h}_s(s)\nu(s) = \boldsymbol{f}_s(s)z(s) + \boldsymbol{G}_s(s)\boldsymbol{u}_s(s), && \text{(12b)}\\
& z'(s) = 2\nu(s), && \text{(12c)}\\
& 0 < z(s) \le \overline{z}, && \text{(12d)}\\
& \underline{\boldsymbol{u}} \le \boldsymbol{u}_s(s) \le \overline{\boldsymbol{u}}, && \text{(12e)}\\
& \text{for } s \in [0,1],\\
& \int_0^1 \frac{1}{\sqrt{z(s)}}\,\mathrm{d}s \le T && \text{(12f)}
\end{aligned}$$

When reformulating problem (10), we lost the traversal time constraint T; therefore, it is reintroduced as (12f). Intuitively, the solver can minimize the objective cost by reducing z and $\boldsymbol{u}_s$; however, decreasing them too much will violate the traversal time constraint (12f).

III. Solution Description

We solve problem (12) using trapezoidal collocation [5], [13]. First, we discretize s into a grid of $K+1$ collocation points. Since the system's dynamics are linear in s, we approximate ν as a piecewise-constant function, z as a linear spline, and $\boldsymbol{u}_s$ as a nonlinear spline.

The objective functional (12a) is approximated as

$$\int_0^1 \frac{\|\boldsymbol{u}_s(s)\|_2^2}{\sqrt{z(s)}}\,\mathrm{d}s \approx \sum_{k=0}^{K}\left[\left\|\boldsymbol{u}_s^k\right\|_2^2 \int_{s^k}^{s^{k+1}} \frac{1}{\sqrt{z(s)}}\,\mathrm{d}s\right], \quad (13)$$

where s^k is the value of s at grid point k, and $\boldsymbol{u}_s^k$ is the value of $\boldsymbol{u}_s$ at collocation point s^k. The analytical integral for the second component of (13) is

$$\int_{s^k}^{s^{k+1}} \frac{1}{\sqrt{z(s)}}\,\mathrm{d}s = \frac{2\left(s^{k+1}-s^k\right)}{\sqrt{z^{k+1}}+\sqrt{z^k}},$$

where z^k is the value of z at collocation point s^k.

Given the above formulation, the trajectory optimization problem (12) is approximated as a convex program:

$$\begin{aligned}
\underset{\substack{z^0,\ldots,z^K\\ \boldsymbol{u}_s^0,\ldots,\boldsymbol{u}_s^{K-1}\\ \nu^0,\ldots,\nu^{K-1}}}{\text{minimize}} \quad & \sum_{k=0}^{K-1} \frac{2\left(s^{k+1}-s^k\right)\left\|\boldsymbol{u}_s^k\right\|_2^2}{\sqrt{z^{k+1}}+\sqrt{z^k}} && \text{(14a)}\\
\text{subject to} \quad & \boldsymbol{h}_s(s^k)\nu^k = \boldsymbol{f}_s(s^k)z^k + \boldsymbol{G}_s(s^k)\boldsymbol{u}_s^k, && \text{(14b)}\\
& z^{k+1} - z^k = 2\nu^k\left(s^{k+1}-s^k\right), && \text{(14c)}\\
& \underline{\boldsymbol{u}}_s \le \boldsymbol{u}_s^k \le \overline{\boldsymbol{u}}_s, && \text{(14d)}\\
& 0 \le z^k \le \overline{z}, && \text{(14e)}\\
& \text{for } k = 0,1,\ldots,K-1,\\
& z^0 = \dot{s}_0^2, && \text{(14f)}\\
& z^K = \dot{s}_1^2, && \text{(14g)}\\
& \sum_{k=0}^{K-1} \frac{2\left(s^{k+1}-s^k\right)}{\sqrt{z^{k+1}}+\sqrt{z^k}} \le T && \text{(14h)}
\end{aligned}$$

A. Second-Order Cone Reformulation

We reduce program (14) into a more efficient, second-order cone program (SOCP) using the reformulation method described in [5]. The reformulation also provides implementation improvement. Convex program solvers that rely on disciplined convex programming, such as CVXPY, may fail to verify the convexity of complicated expressions.[2] The general SOCP structure requires a linear objective function, affine equality constraints, and second-order cone inequality constraints [14, Sec. 4.4.2].

Reformulating the convex program into an SOCP requires the introduction of additional decision variables $a^0, a^1, \ldots, a^K$; $b^0, b^1, \ldots, b^{K-1}$; and $c^0, c^1, \ldots, c^{K-1}$.

We introduce constraints

$$a^k \le \sqrt{z^k} \quad \text{for } k = 0,1,\ldots,K. \quad (15)$$

Then, we re-express objective function (14a) as

$$\sum_{k=0}^{K-1} 2\left(s^{k+1}-s^k\right) b^k$$

and introduce constraints

$$\frac{\left\|\boldsymbol{u}_s^k\right\|_2^2}{a^{k+1}+a^k} \le b^k \quad \text{for } k = 0,1,\ldots,K-1. \quad (16)$$

Similarly, introducing constraints

$$\frac{1}{a^{k+1}+a^k} \le c^k \quad \text{for } k = 0,1,\ldots,K-1, \quad (17)$$

allows us to express (14h) as

$$2\sum_{k=0}^{K-1}\left(s^{k+1}-s^k\right)c^k \le T. \quad (18)$$

Constraints (15) and (16)–(18) will become active as the program converges to a solution. When this happens, the inequalities will become equalities, and the SOCP will resemble the convex program (14).

Constraint (15) is expressed in its conic form with

$$\left\| \begin{matrix} 2a^k \\ z^k - 1 \end{matrix} \right\|_2 \le z^k + 1; \quad (19)$$

constraint (16) becomes

$$\left\| \begin{matrix} 2u_{s,\mathrm{lin}}^k \\ 2u_{s,\mathrm{ang}}^k \\ a^{k+1}+a^k-b^k \end{matrix} \right\|_2 \le a^{k+1}+a^k+b^k; \quad (20)$$

and (17) is

$$\left\| \begin{matrix} 2 \\ a^{k+1}+a^k-c^k \end{matrix} \right\|_2 \le a^{k+1}+a^k+c^k. \quad (21)$$

The components in the left side of inequalities (19)–(21) form column vectors, and $\|\cdot\|_2$ represents the Euclidean norm of those vectors.

[2] https://web.archive.org/web/20211018022509/https://www.cvxpy.org/tutorial/dcp/index.html

Finally, the SOCP reformulation of (14) is given as:

$$\min_{\substack{z^0,\ldots,z^K \\ \boldsymbol{u}_s^0,\ldots,\boldsymbol{u}_s^{K-1} \\ \nu^0,\ldots,\nu^{K-1} \\ a^0,\ldots,a^K \\ b^0,\ldots,b^{K-1} \\ c^0,\ldots,c^{K-1}}} \quad \sum_{k=0}^{K-1} 2\left(s^{k+1}-s^k\right) b^k \tag{22a}$$

$$\text{s.t.} \quad \boldsymbol{h}_s(s^k)\nu^k = \boldsymbol{f}_s(s^k)z^k + \boldsymbol{G}_s(s^k)\boldsymbol{u}_s^k, \tag{22b}$$

$$\underline{\boldsymbol{u}}_s \le \boldsymbol{u}_s^k \le \overline{\boldsymbol{u}}_s, \tag{22c}$$

$$0 \le z^k \le \overline{z}, \tag{22d}$$

$$\left\| \begin{matrix} 2a^k \\ z^k - 1 \end{matrix} \right\|_2 \le z^k + 1, \tag{22e}$$

$$\text{for } k = 0, 1, \ldots, K,$$

$$z^{k+1} - z^k = 2\nu^k \left(s^{k+1} - s^k\right), \tag{22f}$$

$$\left\| \begin{matrix} 2u_{s,\text{lin}}^k \\ 2u_{s,\text{ang}}^k \\ a^{k+1} + a^k - b^k \end{matrix} \right\|_2 \le a^{k+1} + a^k + b^k, \tag{22g}$$

$$\left\| \begin{matrix} 2 \\ a^{k+1} + a^k - c^k \end{matrix} \right\|_2 \le a^{k+1} + a^k + c^k, \tag{22h}$$

$$\text{for } k = 0, 1, \ldots, K-1,$$

$$2\sum_{k=0}^{K-1} \left(s^{k+1} - s^k\right) c^k \le T, \tag{22i}$$

$$z^0 = \dot{s}_0^2, \tag{22j}$$

$$z^K = \dot{s}_1^2 \tag{22k}$$

Some combinations of reference path, traversal time, and control bounds may render this problem infeasible. We assume the reference path is dynamically feasible and that the planner implementation will report if the problem is infeasible. To reduce computation time, a practitioner may place a time-optimal module in front of our system to determine the minimum feasible traversal time.

IV. Experiments

We evaluated our solution against three different path types that vehicles commonly encounter at intersections: a left turn, a right turn, and a straight path. These paths are represented as follows:

$$\tau_{\text{left}}(s) = \begin{bmatrix} \alpha_{\text{left}} \cos(\pi s/2) - \alpha_{\text{left}} \\ \alpha_{\text{left}} \sin(\pi s/2) \\ \arctan\left(\tau'_{\text{left},2}(s)/\tau'_{\text{left},1}(s)\right) \end{bmatrix}$$

$$\tau_{\text{right}}(s) = \begin{bmatrix} -\alpha_{\text{right}} \cos(\pi s/2) \\ \alpha_{\text{right}} \sin(\pi s/2) \\ \arctan\left(\tau'_{\text{right},2}(s)/\tau'_{\text{right},1}(s)\right) \end{bmatrix}$$

$$\tau_{\text{straight}}(s) = \begin{bmatrix} 1 \\ \alpha_{\text{straight}} s \\ \arctan\left(\tau'_{\text{straight},2}(s)/\tau'_{\text{straight},1}(s)\right) \end{bmatrix}$$

Parameters α_{left}, α_{right}, and α_{straight} represent the arc radius (for turns) and length (for straight paths). Table I shows the path parameters, traversal time ranges, and actuator limits used in our evaluations.

TABLE I
Simulation Parameters

Parameter	Min. Value	Max. Value	Step	Unit
α_{left} and α_{right}	5	15	1	m
α_{straight}	5	15	1	m
$\overline{\boldsymbol{u}}_{t,\text{lin}}$ and $\overline{\boldsymbol{u}}_{s,\text{lin}}$	-	2.5	-	m/s^2
$\underline{\boldsymbol{u}}_{t,\text{lin}}$ and $\underline{\boldsymbol{u}}_{s,\text{lin}}$	-2.5	-	-	m/s^2
$\overline{\boldsymbol{u}}_{t,\text{ang}}$ and $\overline{\boldsymbol{u}}_{s,\text{ang}}$	-	2.5	-	rad/s^2
$\underline{\boldsymbol{u}}_{t,\text{ang}}$ and $\underline{\boldsymbol{u}}_{s,\text{ang}}$	-2.5	-	-	rad/s^2
T	5	25	1	s

TABLE II
Traversal Time Error Metrics

Path	Mean Error	Standard Deviation	Runs
τ_{left}	6.7501×10^{-8}	8.6470×10^{-8}	231
τ_{right}	6.8975×10^{-8}	9.0152×10^{-8}	231
τ_{straight}	1.1068×10^{-7}	1.0731×10^{-7}	231
All	8.2594×10^{-8}	9.7135×10^{-8}	693

Each run is a combination of specific $\alpha_\square$ and T values.

Our solution is implemented in Python using CasADi [15] to generate the path and its derivatives and CVXPY [16] to model and solve the SOCP problem. The code for our experiments is available in our lab's GitHub repository.[3]

For the collocation implementation, we performed a hyperparameter sweep over $K \in [5, 200]$, the grid resolution, and found that $K = 20$ provided both reasonable approximation and solver stability.

We simulated the vehicle using a phase space variant of the second-order unicycle [3, eq. (13.46)]. The model's state vector $\boldsymbol{\xi} \in \mathbb{R}^5$ is $\boldsymbol{\xi} \coloneqq (x, y, \theta, v, \omega)$, and its state transition is defined by

$$\dot{x} = v\cos\theta \quad \dot{y} = v\sin\theta \quad \dot{\theta} = \omega$$
$$\dot{v} = u_{t,\text{lin}} \quad \dot{\omega} = u_{t,\text{ang}}$$

After solving SOCP problem (22), the system control is approximated with a third-order spline using cubic interpolation. This interpolated control function is denoted $\hat{\boldsymbol{u}}_s \colon [0,1] \to \mathbb{R}^2$. We also approximated the transformed differential state using the same interpolation method and refer to the interpolant as $\hat{z} \colon [0,1] \to \mathbb{R}$.

The simulated robot was actuated using the control signal generated by the motion planner. We created a grid over s using $J+1$ points and integrated the system over the grid point intervals using Runge–Kutta fourth-order (RK4) integration. The control signal $\hat{\boldsymbol{u}}_s$ is held constant within each interval $[s^j, s^{j+1}]$, for $j = 0, 1, \ldots, J-1$.

A. Experimental Results

Table II shows the mean traversal time error and standard deviation for all three path types over the different path parameters and traversal times. As shown in the table, the robot's traversal durations matched its assigned times and with minimal error.

Fig. 2 visualizes the simulated robot's trace for one of the test paths. Fig. 3 visualizes the system trajectory and control

[3]https://github.com/the-hive-lab/trajectory_optimization

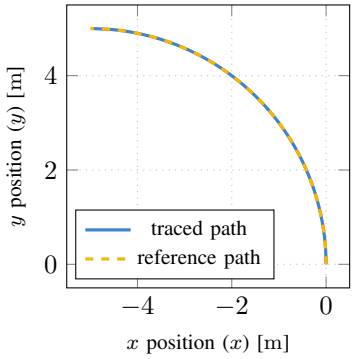

Fig. 2. Simulated robot's path trace. The dashed gold line represents a left-turn reference path, and the solid blue line is the robot's path.

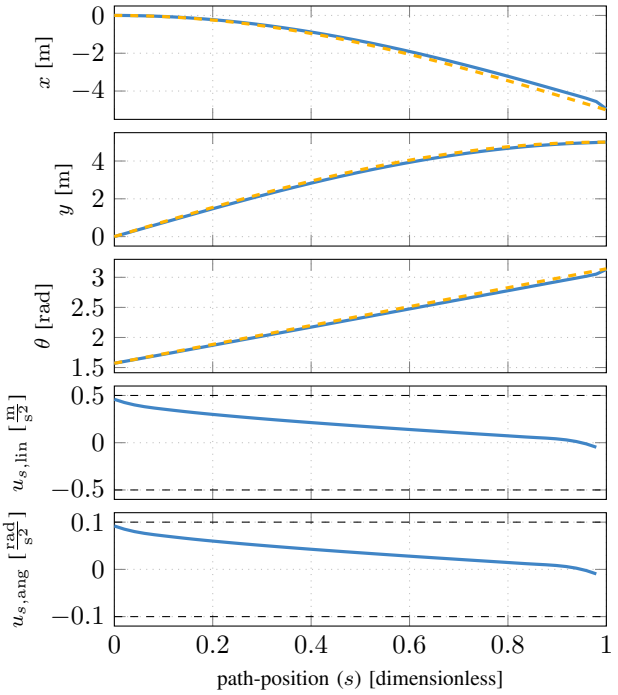

Fig. 3. State trajectory from an example simulation. The dashed gold lines are the reference components of the path. The solid blue lines are the system's trace from the simulation.

inputs for the same test path. Due to space constraints, we omit the scalar linear and angular velocity plots. The first three plots show a slight tracking error, which we believe is due to accumulated approximation errors in the simulation.

The experiment computer was equipped with an Intel Core i5-8250U processor and 16 GB of memory. The solver runtime across all runs averaged 27.44 ms with a standard deviation of 51.86 ms, suggesting it could be performed in online.

V. Conclusion

In this paper, we proposed a convex optimization solution to the control-minimal time-assigned path-constrained trajectory optimization problem. Using a nonlinear change of variables, the original problem formulation is converted into a convex problem then solved using trapezoidal collocation. From the experimental results, we conclude that our proposed method works as expected. The robot traversed the given paths in the specified time and tracked it with minimal error.

One limitation to our approach is that it may not extend to other motion models. Expressing the robot's dynamics in a form that was linear in the squared path-velocity and the path-acceleration was critical to this approach, but other motion models may not have this structure. Another limitation is that we assume a collision-free reference path. If an obstacle blocks the path, a higher-level path planner will have to find a new one. Fortunately, our runtime is low enough that this re-planning process could be performed online.

For future work, we plan to apply our approach to other motion models, such as the second-order bicycle model. Additionally, we will integrate our planner into the Robot Operating System (ROS) 2 middleware to simulate it in the Gazebo simulator and implement it on a physical robot.

References

[1] Z. Zhong, M. Nejad, and E. E. Lee, II, "Autonomous and semiautonomous intersection management: A survey," *IEEE Intell. Transp. Syst. Mag.*, vol. 13, no. 2, pp. 53–70, 2021.

[2] A. Morrissett, P. J. Martin, and S. Abdelwahed, "Socially-optimal auction-theoretic intersection management system," in *2022 IEEE Intell. Veh. Symp. (IV)*, June 2022, pp. 1340–1346.

[3] S. M. LaValle, *Planning Algorithms*. New York, USA: Cambridge University Press, 2006.

[4] H. Pham and Q.-C. Pham, "A new approach to time-optimal path parameterization based on reachability analysis," *IEEE Trans. Robot.*, vol. 34, no. 3, pp. 645–659, June 2018.

[5] D. Verscheure, B. Demeulenaere, J. Swevers, J. D. Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Trans. Autom. Control*, vol. 54, no. 10, pp. 2318–2327, October 2009.

[6] Q.-C. Pham and O. Stasse, "Time-optimal path parameterization for redundantly actuated robots: A numerical integration approach," *IEEE/ASME Trans. Mechatron.*, vol. 20, no. 6, pp. 3257–3263, December 2015.

[7] P. Shen, H. Zou, X. Zhang, Y. Li, and Y. Fang, "Platoon of autonomous vehicles with rear-end collision avoidance through time-optimal path-constrained trajectory planning," in *2017 11th Int. Workshop on Robot Motion and Control (RoMoCo)*, July 2017, pp. 232–237.

[8] P. Shen, X. Zhang, Y. Fang, and M. Yuan, "Real-time acceleration-continuous path-constrained trajectory planning with built-in tradeoff between cruise and time-optimal motions," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 4, pp. 1911–1924, October 2020.

[9] B. Gopalakrishnan, A. K. Singh, and K. M. Krishna, "Time scaled collision cone based trajectory optimization approach for reactive planning in dynamic environments," in *2014 IEEE/RSJ Int. Conf. Intell. Robot. and Syst.*, September 2014, pp. 4169–4176.

[10] A. K. Singh and Q.-C. Pham, "Reactive path coordination based on time-scaled collision cone," *J. Guid. Control, and Dynamics*, vol. 41, no. 9, pp. 2031–2038, September 2018.

[11] S. Frölander and R. Hasan, "Convex optimization-based design of a speed planner for autonomous heavy duty vehicles," Master's thesis, KTH Royal Institute of Technology, Stockholm, SE, 2019.

[12] C. Guarino Lo Bianco and M. Romano, "Optimal velocity planning for autonomous vehicles considering curvature constraints," in *Proc. 2007 IEEE Int. Conf. on Robot. and Automat.*, April 2007, pp. 2706–2711.

[13] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Rev.*, vol. 59, no. 4, pp. 849–904, 2017.

[14] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, USA: Cambridge University Press, 2004.

[15] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Math. Program. Comput.*, vol. 11, pp. 1–36, 2019.

[16] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *J. Mach. Learn. Res.*, vol. 17, no. 83, pp. 1–5, 2016.