Proceedings of the
47th IEEE Conference on Decision and Control
Cancun, Mexico, Dec. 9-11, 2008

On-line Path Planning For an Autonomous Vehicle in an Obstacle Filled Environment

Jeremy D. Schwartz and Mark Milam

Abstract— **The 2007 Darpa Urban Challenge called for a kinematic vehicle path planning method that could navigate and park in an obstacle-filled environment with realistic vehicle constraints. Key capabilities include minimum-time trajectories with both forward and reverse segments, and obstacle avoidance. An algorithm which generalizes the collocation method is developed and used to optimally control a differentially flat parameterization of the kinematic car. The singularity inherent in the differentially flat formulation is addressed without any constraints imposed on the state space. Experimental data is presented showing this algorithm running on Alice, the Caltech autonomous vehicle. The flexibility of the algorithm developed in this paper allows it to be applied to a large group of practical optimal control problems.**

I. Prior Work

The problem of vehicle path planning has been widely studied. Several different methods have been developed that can get a car-like model from point A to point B. Reeds et. al. provide the basis for many methods by proving that the optimal path between any two points (without regard to obstacles) will always flow from one of 48 general constructions [1]. Based on this work, Laumond et. al. develop a metric based on the shortest possible path for a car-like robot [2]. Panizza et. al. provide a method for planning a path with minimum maneuvers (a change of direction defines a new maneuver) along a narrow road of unknown curvature [6]. Müller et. al. develop a method for determining trajectories specifically for the parallel parking problem [3].

The concept of using collocation to solve an optimal control problem by turning it into a nonlinear programming problem has also been studied extensively. Hargraves et. al. developed this technique and applied it specifically to trajectory optimization [7]. This technique has been applied to a wide variety of problems. Von Stryk et. al. developed a specialized form of collocation that merged the standard direct collocation method with an indirect multiple shooting method to improve accuracy [8].

II. Introduction

A new algorithm is developed and implemented to solve the general problem of optimal path planning for a kinematic car in the presence of obstacles. It is then tested in a variety of situations, both in simulation and on Alice, the Caltech autonomous vehicle and entry to the 2007 Darpa Urban Challenge.

Northrop Grumman Space Technology, Redondo Beach, CA, 90278, United States

Fig. 1. Picture of Alice, the Caltech Autonomous Vehicle

This algorithm is an on–line path planning algorithm which computes optimal trajectories in real time. It incorporates both linear constraints, such as boundary polynomials, and nonlinear non-convex constraints, such as obstacle ellipsoids. It also handles higher order constraints on the problem variables, such as vehicle steering angle and steering rate constraints.

The algorithm is an enhancement of the Nonlinear Trajectory Generation (NTG) algorithm. This algorithm, denoted OTG (Optimal Trajectory Generation), improves upon NTG by changing the independent variable set that is used for the problem formulation. NTG used the coefficients of the interpolated B-splines as its independent variable set. OTG instead uses the sampled values of the flat variables and their derivatives as its independent variables. This modification improves ease of use, because constraints can be framed in terms of the output variables instead of B-spline coefficients. It also increases on-line computation speed for two reasons: first, a larger portion of the computation can be calculated offline; second, the new formulation is able to make use of the SNOPT solver, which is more computationally efficient than the NPSOL solver used by NTG.

In this paper, a differentially flat parameterization of a kinematic car model is utilized. This ensures that the algorithm produces exact solution trajectories, with zero integration error between the solution control values and the predicted paths.

This parameterization introduces problems related to a singularity when the car's velocity nears zero. The effects of this singularity are discussed, and an effective technique for solving multi-maneuver trajectories in light of the singularity is introduced and implemented.

Several versions of the algorithm are developed to separately handle paths that may require both forward and

978-1-4244-3124-3/08/$25.00 ©2008 IEEE

reverse maneuvers. Solutions that involve one, two, or three different maneuvers are mathematically distinct from each other when using the flat parameterization. Handling logic is implemented which calls each solver in turn and has contingencies if no good direct solution is found.

Finally, test data is presented that demonstrates the algorithm running on a fully autonomous car. Multi-segment planning and obstacle avoidance are demonstrated in a real-world environment.

III. Optimal Trajectory Generation Algorithm

The algorithm designed in this paper, called OTG, is a modification of the Nonlinear Trajectory Generation (NTG) algorithm. As described in [9], the NTG algorithm is designed to solve an Optimal Control Problem (OCP) by transforming it into a NonLinear Programming problem (NLP) and solve that NLP using a Sequential Quadratic Programming (SQP) solver – in that case, the NPSOL solver. OTG uses the same technique, but because of the change of variable set, it is able to generate a sparse NLP, and thus make use of SNOPT instead of NPSOL. SNOPT is specially designed to solve sparse NLP's, and is computationally faster than the more general NPSOL.

A classical (nonlinear) OCP takes the following form:

$$\begin{aligned} \min_{u} \quad & \int_0^T f(\mathbf{x}(t),\mathbf{u}(t))\ d\tau + g(x(T)) \qquad (1)\\ \text{subject to:} \quad & \dot{\mathbf{x}}(t) = g(\mathbf{x}(t),\mathbf{u}(t)),\\ & \mathbf{x}(0) = \mathbf{x_0}\\ & lb_0 \le \psi(\mathbf{x}(0),\mathbf{u}(0)) \le ub_0,\\ & lb_t \le S(\mathbf{x}(t),\mathbf{u}(t)) \le ub_t. \end{aligned}$$

OTG solves the OCP with three steps. First, a variable or set of variables which represent the OCP states and controls is found. The constraints and cost functions of the OCP are reframed in terms of this variable set. Second, a discrete time sampling is taken of these continuous variables and their derivatives, forming a new set of independent variables. Third, one or more B-spline interpolation curves are created which are fully defined by this independent variable set.

To accomplish the first step, a variable or set of variables $\mathbf{z}$ must be defined

$$\mathbf{z} = A(\mathbf{x},\mathbf{u},\mathbf{u}^{(1)},\ldots,\mathbf{u}^{(s)}) \qquad (2)$$

such that $(\mathbf{x}(t),\mathbf{u}(t))$ can be determined completely from

$$(\mathbf{x},\mathbf{u}) = F(\mathbf{z},\mathbf{z}^{(1)},\ldots,\mathbf{z}^{(s)}) \qquad (3)$$

and the OCP dynamic constraints can be determined by

$$0 = c(\mathbf{z},\mathbf{z}^{(1)},\ldots,\mathbf{z}^{(s)}) \qquad (4)$$

where $\mathbf{z}^{(i)}$ denotes the ith time derivative of $\mathbf{z}$. Note that if a flat parameterization [4] is used for $\mathbf{z}$, then the problem is translated into a lower–dimensional space, and Equation (4) is satisfied by definition, and does not need to be enforced.

To accomplish the second step, a strictly increasing set of n times (called *breakpoints*) is defined:

$$\{t_0, t_1, ..., t_{n-1}\} \qquad (5)$$

where t_0 represents the initial time for the problem, and $t_{n-1} = t_f$, the final time for the problem. An independent variable set, $\bar{\mathbf{z}}$, can be constructed by taking the associated time samples of $\mathbf{z}$:

$$\begin{aligned} \bar{\mathbf{z}} = \quad & \{\bar{\mathbf{z}}_\mathbf{1}, ..., \bar{\mathbf{z}}_\mathbf{q}\},\ \text{where} \qquad (6)\\ \bar{\mathbf{z}}_\mathbf{j} = \quad & \{z_j(t_0), z_j(t_1), ..., z_j(t_{n-1}), ...,\\ & z_j^{(i_j)}(t_0), z_j^{(i_j)}(t_1), ..., z_j^{(i_j)}(t_{n-1})\} \qquad (7) \end{aligned}$$

where q is the number of decision variables in $\mathbf{z}$, and i_j represents the number of derivatives of the j^{th} variable required to satisfy Equation (3) (including the zeroeth derivative).

In order to enforce constraints at times other than the breakpoints, a B-spline parameterization must be chosen:

$$\begin{aligned} \bar{\mathbf{z}}_\mathbf{1} &= \textstyle\sum_{i=1}^{p_1} \mathbf{B}_{i,k_1}(t) c_i^1\\ \bar{\mathbf{z}}_\mathbf{2} &= \textstyle\sum_{i=1}^{p_2} \mathbf{B}_{i,k_2}(t) c_i^2\\ &\vdots\\ \bar{\mathbf{z}}_\mathbf{q} &= \textstyle\sum_{i=1}^{p_q} \mathbf{B}_{i,k_q}(t) c_i^q\\ &\text{and} \quad p_j = (n-1)(k_j - m_j) + m_j \end{aligned}$$

where $B_{i,k_j}(t)$ is the B-spline basis function defined in de Boor [5] for the output z_j with order k_j, c_i^j are the coefficients of the B-spline, n is the number of breakpoints, and m_j is number of smoothness conditions at the breakpoints. Defining

$$\mathbf{c_j} = [c_1^j\ c_2^j\ c_3^j\ ...\ c_{p_j}^j]^T \qquad (8)$$

$$\mathbf{B_j} = [\mathbf{B}_{1,k_j}\ \mathbf{B}_{2,k_j}\ \mathbf{B}_{3,k_j}\ ...\ \mathbf{B}_{p_j,k_j}]_{k_j n \times p_j} \qquad (9)$$

the above expression can be simplified to:

$$\bar{\mathbf{z}}_\mathbf{j} = \mathbf{B_j}\bar{\mathbf{c}}_\mathbf{j} \qquad (10)$$

For this construction, it is ideal if the relationship between the variable set $\bar{\mathbf{z}}_\mathbf{j}$ and the coefficients $\bar{\mathbf{c}}_\mathbf{j}$ is be one-to-one and onto. This prevents the need to do a least-squares approximation of the coefficients given the flat variables. Therefore, $\mathbf{B_j}$ should be square and invertible.

Recall that the entire NLP variable set is denoted $\bar{\mathbf{z}}_\mathbf{j}$, and its size is $i_j n \times 1$, where n is the number of breakpoints and i_j is the number of variable derivatives included for z_j at each breakpoint (including the zeroeth). The coefficients for the interpolation B-splines are denoted $\bar{\mathbf{c}}_\mathbf{j}$, which has size $1 \times (n-1)(k_j - m_j) + m_j$, where k_j is the B-spline polynomial order and m_j is the B-spline smoothness. In order to ensure that $\mathbf{B_j}$ is square, the following relationship is required:

$$(n-1)(k_j - m_j) + m_j = i_j n \qquad (11)$$

If the B-spline smoothness is forced to be equal the number of included variable derivatives, and the B-spline polynomial order is forced to be double this number:

$$\begin{aligned} m_j &= i_j \qquad (12)\\ k_j &= 2i_j \end{aligned}$$

then Equation (11) holds for all n and i_j. Indeed, at this point collocation can be seen as a specific instance of OTG,

where $i_j = 2\ \forall j$, and the resulting polynomials are fourth order.

It must be noted that $x(t_i)$ and $x(t_{i+1})$ are separate variables. They are not connected by a time variable, as they are in the OCP. In order to enforce a relationship between them, constraints must be added at additional points, called *enforcement points*. These enforcement points can now be defined:

$$\{t_0, t_{e_1}, ..., t_{e_{p-1}}\} \tag{13}$$

$$\bar{\mathbf{y}} = \{\bar{\mathbf{y}}_\mathbf{1}, ..., \bar{\mathbf{y}}_\mathbf{q}\}, \text{ where} \tag{14}$$

$$\bar{\mathbf{y}}_\mathbf{j} = \{z_j(t_0), ..., z_j(t_{p-1}), ..., z_j^{(i_j)}(t_0), ..., z_j^{(i_j)}(t_{p-1})\} \tag{15}$$

Note that the increasing sequence of times in Equation (13) is different than the sequence of times in Equation (5), but $t_{e_{p-1}} = t_f$ as before. Since all constraints must be placed in terms of the independant variable set $\bar{\mathbf{z}}$, the values of $z(t_{e_j})$ must be defined in terms of $\bar{\mathbf{z}}$.

Utilizing the same coefficients c_j defined in Equation (8), a new matrix B_{y_j} can be defined such that

$$\bar{\mathbf{y}}_\mathbf{j} = \mathbf{B}_{\mathbf{y_j}}\bar{\mathbf{c}}_\mathbf{j}\ \forall j \in \{1, ..., q\} \tag{16}$$

Since $\mathbf{B_j}$ is invertible, Equation (10) can be expressed:

$$\bar{\mathbf{c}}_\mathbf{j} = \mathbf{B}_\mathbf{j}^{-\mathbf{1}}\bar{\mathbf{z}}_\mathbf{j} \tag{17}$$

From Equations 16 and 17, it can be seen that all of the enforcement points can be put in terms of the independent variable set:

$$\bar{\mathbf{y}}_\mathbf{j} = \mathbf{B}_{\mathbf{y_j}}\mathbf{B}_\mathbf{j}^{-\mathbf{1}}\bar{\mathbf{z}}_\mathbf{j} \tag{18}$$

Then the problem can be stated as an NLP:

$$\min_{\bar{\mathbf{z}} \in R^M} F(\bar{\mathbf{z}}) \qquad \text{subject to} \quad lb \leq c(\bar{\mathbf{y}}(\bar{\mathbf{z}})) \leq ub \tag{19}$$

where M represents the dimension of $\bar{\mathbf{z}}$ and $c(\bar{\mathbf{y}}(\bar{\mathbf{z}}))$ represents the constraints from the OCP, rewritten in terms of the new variable set.

It is worth noting that all of the parameters that define $\mathbf{B_j}$ and $\mathbf{B_{y_j}}\ \forall j \in \{1, ..., q\}$ are selected off-line for any particular problem, and are constant during on-line computation. Thus these matrices can be calculated off-line to improve on-line computation speed. This fact outlines the advantage and major distinction of OTG from NTG. Under the NTG algorithm, the independent variable set for the NLP is the set of B-spline coefficients, $\bar{\mathbf{c}}$. OTG, using the construction above, utilizes the sampled variables and derivatives $\bar{\mathbf{z}}$ as its independent variable set, and this change allows more of the computation to be pushed off-line.

A. Collocation as an instance of OTG

In the simplest case, the OCP state variables $\mathbf{x}$ and control variables $\mathbf{u}$ themselves can be used as $\mathbf{z}$, and the set of NLP state variables looks like

$$\bar{\mathbf{z}} = \{\mathbf{x}(t_0), ..., \mathbf{x}(t_n), \dot{\mathbf{x}}(t_0), ..., \dot{\mathbf{x}}(t_n), \mathbf{u}(t_0), ..., \mathbf{u}(t_n)\} \tag{20}$$

In collocation, following the rules defined in Equation 12, the B-spline polynomials for the state variables must be fourth order, and the control variables have second order polynomials (i.e. linear interpolations between breakpoints). In order to enforce the relational constraints described above, standard collocation necessitates exactly one enforcement point on each B-spline curve in between the breakpoints. The midpoint of each curve (i.e. equidistant between each breakpoint) is usually selected as the enforcement point for symmetry and ease of derivation.

For a more detailed description of collocation, see [7].

B. Integration Error with Collocation

Collocation attempts to force several variables, all approximated with B-spline polynomial curves, to obey a dynamic constraint equation along their entire continuous length. Since dynamic systems generally do not have polynomial solutions, there is inherent residual approximation error. A fourth-order polynomial will never be able to exactly reproduce higher-order behavior, or behavior that does not have a polynomial basis, such as an exponential or a sinusoid. If a collocation solution's control values are used to propagate the model from the initial state, then the resulting path does not exactly match the solution path.

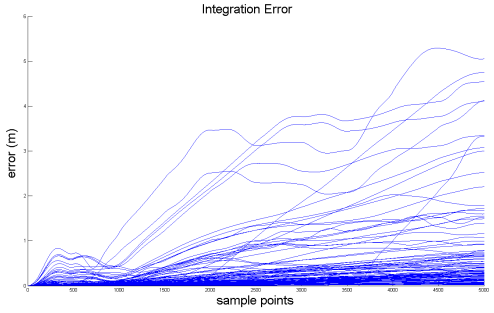

Fig. 2. Integration error created by propagating the collocation solution control values.

Figure 2 shows the integrated error from many test runs of the same collocation algorithm with different start and end points. This version of collocation used 11 breakpoints, necessitating 10 polynomials and 21 enforcement points. The error is zero at all the enforcement points, but since the dynamic constraints for a kinematic car are nonlinear, the error is nonzero in between these points.

The major advantage of the flat formulation is that dynamic constraints are satisfied by definition. This is because the original state and control variables are *defined* using derivatives of the flat variables [4]. The flat formulation of OTG does not define any separate curves for the original state and control — rather, they are reconstructed from the flat variables using Equation (3) after the solution has been determined.

Note that since the flat formulation also uses polynomial basis functions, the same polynomial approximation prevents OTG from producing exactly optimal solutions. Still, though slightly suboptimal, if the derived control variables are taken

from an OTG solution and propagated in a feed-forward manner using the dynamic constraints, then they will match the solved trajectories with **zero** integration error. Contrast this with Figure 2.

IV. Modelling Caltech's Autonomous Vehicle

A. Optimization Objectives

The 2007 Darpa Grand Challenge was to develop an autonomous car that could safely navigate an urban environment. One aspect of this challenge was navigation in **zones**, a term used to describe areas without any sort of lane markings or guiding paths. (A parking lot is a common example of a zone.) In addition to a lack of lane markings, a zone has a boundary, and may contain both moving and stationary obstacles. All of these obstacles must be avoided while the car executes a maneuver — such as parking — within the zone in minimum time.

B. Kinematic Car Model

Alice is modelled as a standard kinematic car [1]. This model has four state variables: x and y position, speed v, and heading angle θ. In addition, there are two input variables: acceleration a, and the steering angle ϕ. The dynamic equations are:

$$\begin{array}{rcl} \dot{x}(t) &=& v(t)\cos(\theta(t)) \\ \dot{y}(t) &=& v(t)\sin(\theta(t)) \\ \dot{v}(t) &=& a(t) \\ \dot{\theta}(t) &=& \frac{v(t)}{L}\tan(\phi(t)) \end{array} \tag{21}$$

where L represents the distance between the front and rear axle.

There are several constraints in this problem. First, there are upper and lower bounds on velocity, acceleration, and steering angle:

$$\begin{array}{rcl} v_{min} &<& v(t) < v_{max} \\ a_{min} &<& a(t) < a_{max} \\ \phi_{min} &<& \phi(t) < \phi_{max} \end{array} \tag{22}$$

There are also keep-out constraints which represent obstacles, and keep-in constraints which represent a bounding zone. In this problem, the keep-out constraints are formed as ellipses. Given an obstacle location of (h, k), a height a and a width b, the constraint takes form of a standard ellipse equation:

$$(\frac{x(t)-h}{a})^2 + (\frac{y(t)-k}{b})^2 > 1 \ \forall t \in [0, T] \tag{23}$$

The keep-in constraints are simple linear bounds of the form $cy < mx+b$, or $cx < b$ for vertical lines. The objective of this problem is to minimize time.

C. Defining flat variables

x and y can be used as the flat variables for this problem. Using these variables and their derivatives, the other state variables can be redefined:

$$\begin{array}{rcl} v(t) &=& \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2} \\ \theta(t) &=& \arctan\frac{\dot{y}(t)}{\dot{x}(t)} \end{array} \tag{24}$$

The control inputs must also be redefined in terms of the flat variables:

$$\begin{array}{rcl} a(t) &=& \frac{\dot{x}\ddot{x}+\dot{y}\ddot{y}}{\sqrt{\dot{x}(t)^2+\dot{y}(t)^2}} \\ \phi(t) &=& \arctan(\frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{(\dot{x}^2+\dot{y}^2)^{\frac{3}{2}}}L) \end{array} \tag{25}$$

In order to optimize time, the problem is framed in normalized timespace, and a time scale factor ξ is used. This time scale variable is an additional decision variable, and it is derived from the equation $\xi\tau = t$, where τ ranges from 0 to 1. Differentiating this equation gives the relationship:

$$\begin{array}{rcl} \frac{dt}{d\tau} &=& \xi \\ \frac{dx}{dt} &=& \frac{1}{\xi}\frac{dx}{d\tau} \end{array} \tag{26}$$

Incorporating ξ, the flat definitions then become:

$$\begin{array}{rcl} v(t) &=& \frac{1}{\xi}\sqrt{x'(\tau)^2 + y'(\tau)^2} \\ \theta(t) &=& \arctan\frac{y'(\tau)}{x'(\tau)} \\ a(t) &=& \frac{1}{\xi^2}\frac{x'x''+y'y''}{\sqrt{x'^2+y'^2}} \\ \phi(t) &=& \arctan(\frac{x'y''-y'x''}{(x'^2+y'^2)^{\frac{3}{2}}}L) \end{array} \tag{27}$$

Additionally, the cost function can be stated simply as:

$$J(\bar{\mathbf{z}}) = \xi \tag{28}$$

Any boundary constraints on the state variables must be reframed in terms of the flat variables. In this case, the bounds shown in Equation (22) must be rewritten as:

$$\begin{array}{rcl} v_{min} &<& \frac{1}{\xi}\sqrt{x'(\tau)^2 + y'(\tau)^2} < v_{max} \\ a_{min} &<& \frac{1}{\xi^2}\frac{x'x''+y'y''}{\sqrt{x'^2+y'^2}} < a_{max} \\ \phi_{min} &<& \arctan(\frac{x'y''-y'x''}{(x'^2+y'^2)^{\frac{3}{2}}}L) < \phi_{max} \end{array} \tag{29}$$

Notice that as a result of the flat parameterization, these constraints are now nonlinear.

D. Singularity

One significant side effect of the flat formulation in the kinematic car model is a singularity when $v = 0$. In this case, θ becomes undefined, and $\dot{\theta} \to \infty$ as $v \to 0$.

Additionally, by definition, the velocity can never be negative. Indeed, under the flat formulation, changing direction requires a small but important change in the dynamic equations, and thus requires a different model.

It is important to note that under the flat parameterization, the problem where the vehicle is going forward and the problem where the vehicle is travelling in reverse are distinct from each other. Specifically, in the reverse case, θ must

be set to the polar opposite of its value for the forward case. This can be accomplished by adding π to the above definition of θ, or using the quadrant-sensitive version of arctan: $\theta_{rev} = \arctan 2(-\dot{y}(t), -\dot{x}(t))$.

This distinction makes it impossible to calculate a multi-part trajectory using a single path. However, this framework handles multipath trajectories by handling each path segment as a separate set of variables for the NLP to optimize.

E. Implementation

1) Single piece trajectory: The simplest solution which OTG may determine for this problem would be a single-directional path from the initial point to the final point, obeying all dynamic and obstacle constraints while optimizing trajectory travel time. $x(\tau)$, $y(\tau)$, and ξ are used as the basis for the OTG optimization, and the corresponding control values are determined as in Equation (25).

2) Multipath trajectories: OTG can calculate multipath trajectories (trajectories with both forward and reverse segments) if separate path variables are created for each segment, and all segments are optimized at once. In this case, instead of using just x, y, and ξ as the flat variables, the set includes $\{x_1, y_1, \xi_1, ...x_n, y_n, \xi_n\}$ for n path segments. To ensure that the path segments form a complete continuous trajectory, constraints must be added which enforce equality on the connected boundary states of each path:

$$\begin{array}{rcl} x_1|_{\tau=1} & = & x_2(0) \\ y_1|_{\tau=1} & = & y_2(0) \\ v_1|_{\tau=1} & = & v_2(0) = \epsilon > 0 \\ \theta_1|_{\tau=1} & = & \theta_2(0) \end{array} \tag{30}$$

The cost function for this compound problem consists of the sum of all individual costs:

$$J(\bar{\mathbf{z}}) = \xi_1 + \xi_2 + ... + \xi_n \tag{31}$$

With this construction, OTG not only optimizes each path, it optimizes the location of the adjoining points (called 'cusps') as well. Notice, however, that each new cusp increases the complexity of the problem by a factor of n^2, where n is the size of the searchable space. Thus, computation time increases exponentially as cusp points are added.

Note that each multipath trajectory solver is a distinct formulation, since variables must be added to the NLP set in order to formulate the multipath problem.

3) Handling logic: Because the single-piece algorithm runs much faster than the multipath algorithms, it is prudent to attempt a single-path solution before resorting to multipath solutions. In order to manage this, a logic framework was developed that tries the different OTG algorithms sequentially, based on their complexity and their relevance to the situation. The benefit of this approach is that calculation can be terminated as soon as a proper solution to the problem is discovered.

Once a solution path is found, this path is followed until a trigger event occurs. When such an event occurs, then the planner is reset and OTG starts from the beginning of the logic tree. The most important trigger event is the introduction of a new obstacle into the planned path. Thus, if an obstacle moves in an unexpected way and obstructs the planned path, Alice stops and queries OTG for a new solution.

The other common trigger event is a change of direction. Any time Alice reaches the end of a maneuver, rather than simply following the remaining parts of the existing optimal trajectory, Alice asks OTG to resolve the problem from the new position. This design choice was made largely due to the particulars of the Alice path following architecture, which only stores one single-directional maneuver at a time. This design has the positive effect of demonstrating that the OTG solvers obey the principle of optimality. This effect can be seen in the results section.

4) Initial guess: As with many solvers, OTG works by iteratively improving its guess at a solution. As such, it must be provided with an initial guess to any problem. In the case of a problem as complex and nonlinear as Alice, the quality of the initial guess has a significant effect on the speed and success of the algorithm. Experimental evidence shows that this sensitivity to the initial guess increases greatly with multipath trajectories.

For this problem, Reeds–Schepp [1] curves are generated for the initial guess. For each algorithm, the shortest of the Reeds–Schepp curves that has the same character as the solution curve is used: only one-direction Reeds–Schepp curves are allowed for the single-piece solver, only two-directional curves are admitted for the two-path solver, and so on. The time constant for each guess is estimated by assuming a constant velocity along the initial guess curve, and dividing the total distance of the initial guess curve by this velocity.

The curves produced by the Reeds–Shepp algorithm are very good guesses. However, they do not consider constraints. Most importantly, the Reeds–Schepp algorithm disregard the obstacle and boundary constraints. Since OTG considers all of these factors in its optimization, it often finds optimal solutions which are very different from the Reeds–Schepp initial guesses.

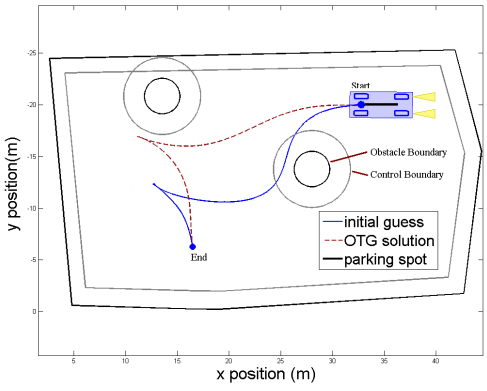

Fig. 3. An initial Reeds–Schepp guess at a path, along with the optimal path solution. This data was generated in simulation.

Figure 3 shows just such a situation. The grey lines around obstacles represent the control boundary on Alice: if the control point on Alice crosses this line, then some part of Alice would contact the obstacle. The paths shown are two-maneuver plans: the first maneuver is a reverse out of a parking spot, and the second maneuver is a forward drive to a predetermined exit point. The initial guess impinges on an obstacle, and would require a very high steering rate in order to execute the sharp S-curve maneuver at constant speed. The OTG solution, which was seeded with the initial guess, determines the optimal trajectory for this scenario.

5) Solution Verification: Since OTG involves a conversion of a continuous-time problem into a discrete set of variables, the problem constraints can only be enforced on a finite number of sample points. In addition, many of the nonlinear constraints may be very high order and irregular. As a result of these two factors, it is often possible to produce solutions where all the constraints are satisfied at the sampled points, but one or more constraints are violated in between the sample points. In the Alice problem, that effect is exaggerated because the car operates very close to the singular region where $v = 0$.

Fortunately, these false positive solutions always fail in a characteristic manner. The velocity dips down to the singular point momentarily, and in that instant, the heading angle flips $180°$. These solutions can be identified by checking the solution trajectory for a spike in $\dot{\theta}$. Since there are constraints on the steering angle, the heading angle has a maximum rate of change. If the solution ever allows $\dot{\theta}$ to move above this threshold, then it is a false solution.

V. Experiments

Using the implementation described above, the algorithm was tested in two different ways: test runs on Alice, and tests in a high fidelity computer simulation of Alice and the environment. All test runs, both simulated and hardware-based, were run in a **zone** as described in section IV.

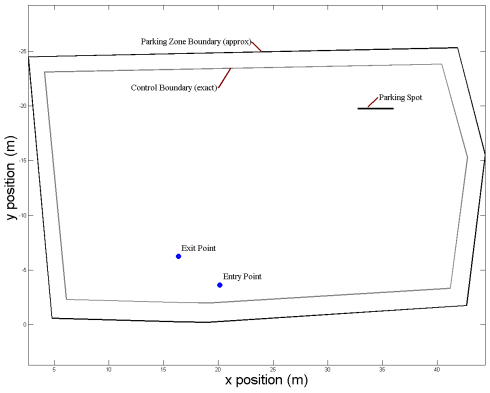

Fig. 4. Baseline layout for all test cases. In every test case, Alice enters the zone and initiates planning at the entry point; pulls forward, properly aligned, into the parking spot; and departs the zone at the exit point.

The overarching task for every test run was to begin on the boundary of the zone, drive to and park in a designated parking spot, and then drive from the parking spot to some designated exit point. The car must park properly in the spot, meaning that in addition to reaching the right position, the car must be correctly oriented to align with the parking spot. Obviously, completing this task would often require several maneuvers.

In these test cases, the same entry point, exit point, and parking spot was used (Figure 4). Since the parking spot is very close to the zone boundary, the only feasible way to exit the parking spot is with a reverse maneuver. As a result, the general solution to the problem is to drive forward from the starting point into the parking spot, back up to a useful intermediate point, and then drive forward to the exit point. Test case 1 is free of obstacles. Test case 2 uses the same zone, but adds several obstacles with which Alice must contend.

Due to the tight turns required in these test cases, and in consideration for the comfort of riders inside Alice during the tests, the maximum velocity for forward maneuvers is $2\frac{m}{s}$. For reverse maneuvers, the maximum velocity is $1\frac{m}{s}$. The maximum steering angle ϕ is 0.45 radians. The maximum acceleration is $0.98\frac{m}{s^2}$, and the maximum breaking is $-3\frac{m}{s^2}$.

VI. Results

All results presented in this section are experimental data, taken fron tests run in real time on Alice.

A. Test Case 1: Baseline

The first test case serves as a baseline to explain the end-to-end trajectory planning process.

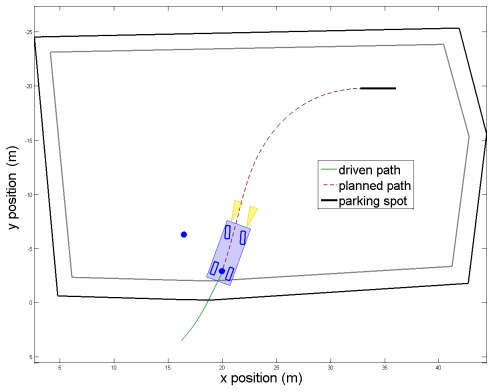

Fig. 5. An early snapshot showing experimental data from test case 1 (zero obstacles).

This process begins when Alice enters the zone. At this point, Alice comes to a halt, and requests a solution path from OTG. A one-piece solution is determined, allowing the car to pull forward into the parking space in a single maneuver (Figure 5).

Once Alice reaches the parking spot, the mission planner (a separate component of the Alice software) determines the appropriate exit point for the zone, and requests a new solution from OTG that will take Alice to the specified exit point. OTG produces a two-piece solution, consisting of a reverse maneuver followed by a forward maneuver. Alice proceeds down the first leg of this solution (Figure 6).

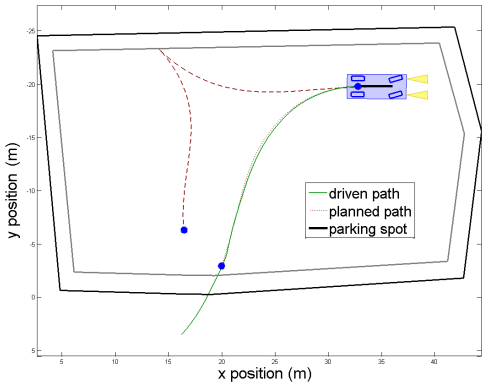

Fig. 6. Completing the parking maneuver in test case 1. A plan to exit the zone is created.

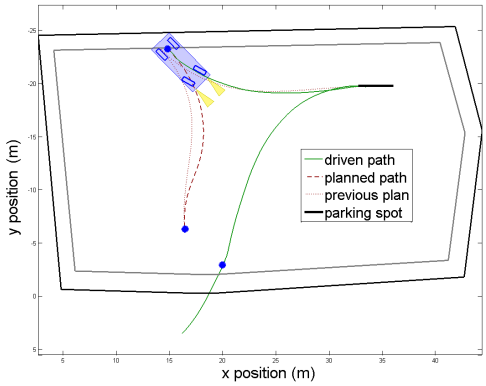

Fig. 7. Alice completes the reverse leg of the 2-piece plan. Alice at the end of each maneuver (change of direction). The new plan is very similar to the second leg of the old plan.

When Alice reaches the end of this maneuver, it again calls for a replan. Due to navigation error and follower inaccuracies, it does not end up exactly where the planned path ended. Still, OTG produces a 1-piece trajectory that is very similar to the second leg of the previous 2-piece trajectory (Figure 7). Finally, Alice follows this new trajectory to the exit point and out of the zone (Figure 8).

Note that when a replan is called for at the end of the reverse maneuver, the resulting 1-segment path is extremely similar to the second part of the 2-segment path. This demonstrates that the OTG solvers obey the basic principle of optimality: if Alice is positioned along an optimal trajectory and a replan is made, then the same trajectory is produced in the replan.

B. Test Case 2: Navigation with Obstacles

The second test case discussed in this paper displays the ability of OTG to dynamically replan around obstacles which are introduced into the environment. This test case begins in the same manner as the first, with Alice entering the zone and pulling in to the parking spot (see figures 10 and 11).

At this point, Alice's sensors pick up some obstacles in the zone. These obstacles are identified with a location, length, and width. This information provides the hard boundary for Alice to avoid. In order to prevent any part of Alice from

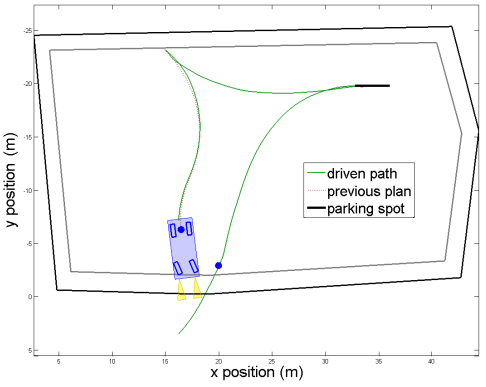

Fig. 8. Alice completes the test case and exits the zone.

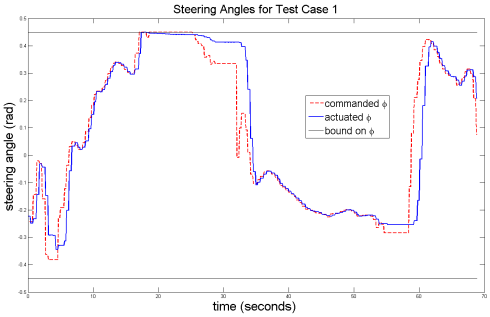

Fig. 9. Commanded and real-world actuated steering angles for test case 1.

contacting obstacles, OTG expands all obstacle boundaries by a $2m$ safety margin for control purposes. The safety margin can be seen drawn around the obstacles. Then, as seen on Figure 11, OTG determines a complete 2-piece solution which is considerably different from the solution seen in test case 1. Note that this solution is only discovered because both legs of the solution are considered at once; the first piece of the solution seen in Figure 6 may not violate any constraints, but Alice would have a hard time leaving the zone from that point.

As before, Alice follows the reverse leg of the two-piece trajectory, and replans when it reaches the end of this maneuver (Figure 12). Again, there is a slight amount of drift, but OTG disregards this because it plans afresh from Alice's new location. In Figure 13, Alice again follows this new path as it exits the zone.

VII. Conclusion

In this paper, a new form of the NTG algorithm, called OTG, was developed. OTG employs a new type of independent variable set which makes it more flexible and increases on-line computation speed. The connection between OTG and collocation was discussed. A differentially flat formulation of the kinematic car was presented and utilized within OTG. As a result, solution paths had no integration error. A singularity that arises as a result of the flat formulation was discussed, and a technique for multi-segment planning was developed.

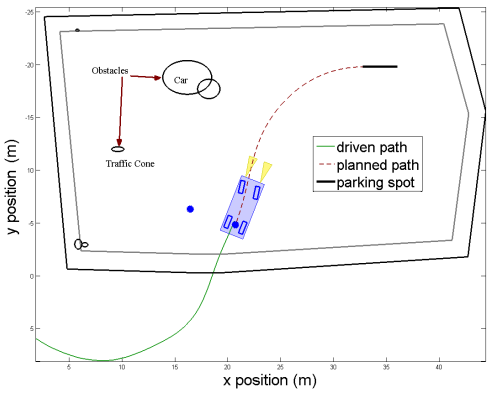

Fig. 10. Test case 2 has the same setup as test case 1, but obstacles have been added to the parking zone.

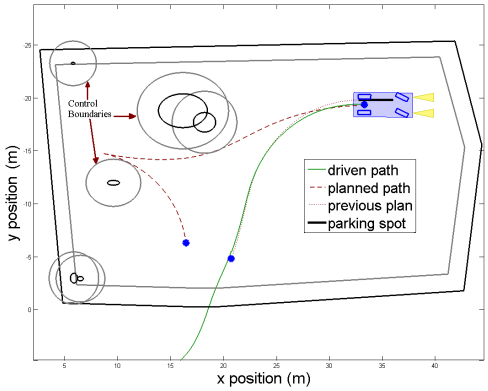

Fig. 11. Alice completes the parking maneuver and detects the obstacles in its way. OTG produces a 2-part trajectory that avoids all obstacles on the way out the zone.

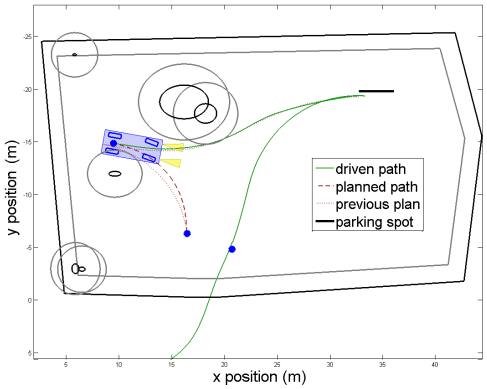

Fig. 12. Alice completes the reverse leg of the 2-part trajectory, and replans at its new location. Alice has drifted off the OTG trajectory slightly, but the new OTG plan handles this because it always starts from Alice's current location.

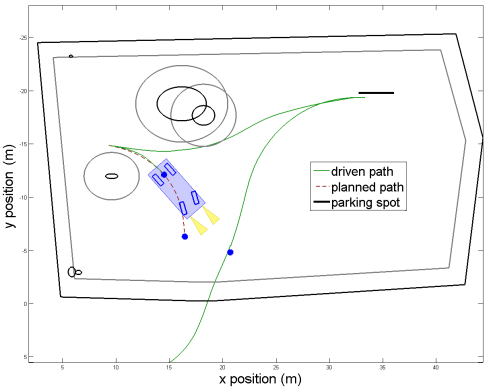

Fig. 13. Alice follows the final OTG trajectory out of the zone.

OTG was implemented and tested on Alice, an autonomous vehicle developed at the California Institute of Technology. OTG displayed the ability to handle complex constraints, obstacle avoidance, multi-segment trajectories, and on-line replanning in a real-world environment.

Future work includes efforts to show additional capabilities of OTG in a comprehensive simulation and hardware-in-the-loop test. The powers of the algorithm developed for this paper extend significantly beyond the capabilities that were possible to test using Alice. For example, this implementation of OTG is fully capable of handling moving obstacles. Also, the speed of OTG coupled with the lack of integration error make it very useful within a feed–forward Receding Horizon Control (RHC) framework [10]. The flexibility of the algorithm developed in this paper also allows it to be applied to a large group of practical optimal control problems.

VIII. Acknowledgements

We would like to acknowledge Richard Murray and Melvin Flores for their help developing and testing the code for OTG and Alice. We would also like to acknowledge Nok Wongpiromsarn for additional help running our tests on Alice.

References

[1] J. A. Reeds and L. A. Schepp, Optimal Paths for a Car That Goes Both Forwards and Backwards, *Pacific Journal of Mathematics*, 1990, Vol. 145 No. 2, pp. 367–393.

[2] J. Laumond, P. Souères, Metric Induced by the Shortest Paths for a Car-like Mobile Robot, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, July 1993, pp. 1299–1303.

[3] B. Müller, J. Deutscher, and S. Grodde, Continuous Curvature Trajectory Design and Feedforward Control for Parking a Car, *IEEE Transactions on Control Systems Technology*, May 2007, Vol. 15 No. 3, pp. 541–553.

[4] M. Fliess, J. Levine, P. Martin, and P. Rouchon, Flatness and Defect of Nonlinear Systems: Introductory Theory and Examples, *International Journal of Control*, 1995, pp. 1327–1360.

[5] C. de Boor, *A Practical Guide To Splines* (Springer-Verlag, 1978).

[6] L. Panizza and R. Frezza, Paths of Bounded Curvature with Minimal Number of Maneuvers, *IEEE Intelligent Vehicles Symposium*, October 2000, pp. 204–209.

[7] C. R. Hargraves and S. W. Paris, Direct Trajectory Optimization Using Nonlinear Programming and Collocation, *AIAA*, July–August 1987, pp. 338–342.

[8] O. von Stryk and R. Bulirsch, Direct and Indirect Methods for Trajectory Optimization, *Annals of Operations Research*, 1992, pp. 357–373.

[9] M. B. Milam, K. Mushambi, and R. M. Murray, A New Computational Approach to Real-Time Trajectory Generation for Constrained Mechanical Systems, *IEEE Conference on Decision and Control*, 2000.

[10] M. B. Milam, R. Franz, J. E. Hauser and R. M. Murray, Receding Horizon Control of Vectored Thrust Flight Experiment, *IEE Proceedings on Control Theory Applications*, May 2005, Vol. 152 No. 3 pp. 340–348.