Optimal Kinodynamic Planning For Autonomous Vehicles

Stavros G. Vougioukas

Agricultural Engineering Laboratory, Faculty of Agriculture
Aristotle University of Thessaloniki
University Campus, P.O. 275, Thessaloniki, Greece

bougis@auth.gr

Abstract – **This paper presents a two-phase kinodynamic planning algorithm, which uses a randomized planner to compute low-cost robotic motions, and optimal control to locally optimize them. The direct transcription approach is used with an NLP numerical optimization algorithm. The algorithms are tested on motion planning for a non-holonomic autonomous vehicle. The results indicate that the two-phase approach is effective in computing optimal motions. However, the dense algebra NLP solver is very time consuming for the optimization of long paths and sparse algebra solvers should be utilized.**

Index Terms – kinodynamic, planning, optimal, direct transcription, NLP.

I. Introduction

Autonomous and semi-autonomous robotic vehicles have already been used in space exploration applications, in military operations, and in entertainment applications (e.g. museum guides). They also hold great promise of entering the areas of services (e.g., health sector) and field machinery operations, with agriculture being a potentially big market. In most applications, the cost of introducing autonomous vehicles could be justified by savings from the optimal execution of the tasks these machines undertake. Optimality may mean faster execution, reduction of fuel costs, minimization of outcome variance, reduction of human health risk, etc. The problem of executing tasks in an optimal manner is nontrivial. In general, it is always coupled with the *optimal motion planning* problem. A motion planning algorithm is said to be complete if it always finds a feasible motion in a finite number of steps, if such a motion exists; otherwise it terminates with an empty solution. The computational complexity of motion planning has been investigated for motion amidst polyhedral obstacles. It has been shown that the problems of finding a feasible or a shortest Euclidean path are both NP-hard. This means that the complexity of such problems scales exponentially in the dimension of the state and the number of obstacles. Hence, complete planners are restricted to solving relatively simple problems, and are not practical for complex real-world applications.

Randomized path planning algorithms have been used successfully to solve high-dimensional path-planning and kinodynamic motion planning problems, which conventional deterministic planners have not been able to handle. In juxtaposition to deterministic planners, which perform a systematic search of the robot's configuration space based on some optimality or heuristic criteria, randomized planners rely on random sampling of the robot's configuration space in order to construct a feasible path. Different sampling schemes have been proposed for efficient coverage of the configuration space, ranging from random walks [1], [2] to random search trees [3], [4], [5] which expand towards unexplored regions of the space. Randomized planners are *probabilistically* complete, which means that in theory, a feasible path will be computed with probability which approaches one as the computation time grows to infinity. However, practice has shown that for real problems a feasible path can be reached in a finite number of iterations, which depends on the complexity of problem.

In many applications an *optimal*, instead of just a feasible motion is required by the planner. The feasible paths computed by the repeated execution of randomized planners may not belong to the same homotopy class. Hence, such paths are not *optimal* in some sense. Furthermore, the paths are not even "smooth" because of the randomness involved in their generation. For PRM-based planners work has been done [6], [7] to select an optimal path from the roadmap graph based on task-dependent optimality criteria, and also to locally improve this path [8], [9] by converting it to a curve, adding or moving nodes from it. This paper presents a two-phase motion planning algorithm which can compute low cost motions, given a desired cost function. The planner is *kinodynamic*, i.e. it computes optimal state trajectories, where a state may contain derivatives of variables (e.g. speed). In the first phase the algorithm utilizes RRT-based [4] randomized planning to compute feasible paths of *monotonically decreasing cost*. In the second phase the motion optimization is formulated within the optimal control framework and a numerical algorithm is used to minimize the cost functional of the *entire* motion. The major contributions of this work are the formulation of a framework for optimal control based smoothing of an RRT produced plan and the extension to the biased bidirectional RRT planner.

II. Problem Statement

Consider the motion of a non-holonomic car-like robot in the plane among static obstacles (Fig. 1).

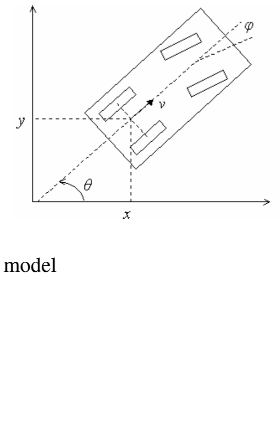

Fig. 1 Car-like robot model

The x and y coordinates give the position of the car's rear-wheel axle midpoint $\mathbf{P} = [x\, y]^T$. The unit vector $\mathbf{v}$ has its origin at $\mathbf{P}$ and lies along the direction of motion. The robot's orientation is given by θ, the angle between the positive x-axis and $\mathbf{v}$. The vehicle's front-wheel turning angle is φ, and its rotational velocity ω, and the linear velocity and acceleration are v and a respectively. The robot state is $\mathbf{x} = [x\, y\, \theta\, \phi\, v]^{\mathrm{T}}$ with dimension $N_x=5$, and the control vector is $\mathbf{u} = [a\, \omega]^{\mathrm{T}}$ with dimension $N_u=2$. The robot's dynamic model is described by a set of differential equations $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u})$ of the form:

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= v\tan\phi / L \\ \dot{\phi} &= \omega \\ \dot{v} &= a \end{aligned} \tag{1}$$

The state and control vectors are subject to constraints of the form

$$\mathbf{x}_{\min} \le \mathbf{x} \le \mathbf{x}_{\max}\, , \mathbf{u}_{\min} \le \mathbf{u} \le \mathbf{u}_{\max}. \tag{2}$$

For maximum flexibility, the geometric descriptions of the robot and the obstacles are assumed to be given as collections of geometric primitives (e.g. triangles). Hence, the analytic expressions for the obstacles are not known. However, it is required that a *minimum distance* function $d(\mathbf{x})$ between the robot and all obstacles can be computed for any robot state. Hence, a state is collision-free when

$$d(\mathbf{x}(t)) \ge 0 \tag{3}$$

Given fixed, collision-free initial and goal states $\mathbf{x}_I$ and $\mathbf{x}_f$ at times t_0 and t_f respectively, and a cost functional $J(\mathbf{x}(),\mathbf{u}(),t_f)$, a feasible control function $\mathbf{u}(t)$ is desired, which results in a low cost collision-free state trajectory $\mathbf{x}(t)$. The cost function expresses the desirable optimization criteria, such as minimum length path, minimum control effort, minimum time, etc.

The problem can be approached under the optimal control framework [10], [11], where in general an optimal path can be computed by the minimization of a Lagrangian cost function of the form

$$J = \int_{t_0}^{t_f} \psi(\mathbf{x},\mathbf{u},t)dt \tag{4}$$

where ψ is a cost function. These cost functions encode the desirable optimization criteria, but may also contain penalty-terms, which enforce the problem constraints.

In the general case, such problems can only be solved numerically. There are two general approaches for their solution [12]. The *indirect* approach uses the Pontryagin minimum principle to minimize the problem's Hamiltonian [13]. The basic problem with such approaches is that the adjoint equations and the transversality conditions for the system have to be computed analytically for each problem. This is not well suited to automated planners, which are required to solve problems for different kind of situations (e.g., kinematic, or dynamic motion equations, car-trailer combination, etc.) Furthermore, when path inequality constraints are present (e.g. obstacles) it may be required to predetermine the sequence of constrained and unconstrained sub-motions to formulate the correct boundary value problems. The direct approaches don't use the maximum principle; instead they cast the optimal control problem into a classical constrained Nonlinear Programming Problem (NLP), for which many numerically efficient procedures exist.

III. DIRECT TRANSCRIPTION OPTIMIZATION

The main idea behind direct transcription algorithms is the following [12]. The state equations are discretized at N time intervals defined by the following time instants:

$$t_0 < t_1 < t_2 \cdots < t_N = t_f \tag{5}$$

The robot's state at step k is $\mathbf{x}_k = [x_k\; y_k\; \theta_k\; \phi_k\; v_k]^{\mathrm{T}}$ and the control vector is $\mathbf{u}_k = [a_k\; \omega_k]^{\mathrm{T}}$. A simple Euler discretization of the state equations is the following:

$$\begin{aligned} x_{k+1} &= x_k + (t_{k+1} - t_k)v_k \cos\theta_k \\ y_{k+1} &= y_k + (t_{k+1} - t_k)v_k \sin\theta_k \\ \theta_{k+1} &= \theta_k + (t_{k+1} - t_k)v_k \tan\phi_k / L \\ \phi_{k+1} &= \phi_k + (t_{k+1} - t_k)\omega_k \\ v_{k+1} &= v_k + (t_{k+1} - t_k)a_k \end{aligned} \tag{6}$$

Then, all variables to be optimized are aggregated into a single vector of dimension $n = N_x(N+1)+ N_uN+N+1$, which constitutes the NLP state vector:

$$\mathbf{X} = \left[\mathbf{u}_0^T\; \mathbf{u}_1^T \cdots \mathbf{u}_{N-1}^T\; \mathbf{x}_0^T\; \mathbf{x}_1^T \cdots \mathbf{x}_N^T\; t_0\; t_1 \cdots t_N\right] \tag{7}$$

The optimization criterion equation (4) can be written in discrete form as

$$J = \Psi(\mathbf{X}) \tag{8}$$

and the optimization problem can be put in the following form:

$$\begin{aligned} \mathbf{X}^* &= \arg\min_{\mathbf{X}} \{\Psi(\mathbf{X}) : \quad \mathbf{X} \in S\} \\ S &= \{\mathbf{X} \in \mathbb{R}^n : \mathbf{x}_{\min} \le \mathbf{x}_k \le \mathbf{x}_{\max}\, , \\ &\qquad \mathbf{u}_{\min} \le \mathbf{u} \le \mathbf{u}_{\max}\, , \\ &\qquad d(\mathbf{x}_k) \ge 0\, , \\ &\qquad t_0 < t_1 < t_2 \cdots < t_N = t_f\, , \\ &\qquad \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)\, , k = 0,1,...,N \\ &\qquad \mathbf{x}_0 = \mathbf{x}_I\, , \mathbf{x}_N = \mathbf{x}_f\} \end{aligned} \tag{9}$$

The set S is the constraint set defined by the set of equations (2, 3, 5, 6) together with the fixed initial and final state constraints. Equations (2) is a set of $N_x(N+1)+ N_uN$ linear inequalities. Equations (3) (discretized) are $N+1$ nonlinear inequalities, and Equations (5) $N+1$ linear inequalities. The state equations (6) constitute a set of $N_x N$ linear equality constraints.

The method and numerical procedure used to solve the resulting NLP problem is taken from [14], [15]. The method implemented is a sequential equality constrained quadratic programming method with an active set technique. The active set is estimated using an error criterion for the Kuhn-Tucker-conditions, which is purely local. If linearly dependent gradients of active constraints occur, then the code switches to an alternative usage of a fully regularized mixed constrained sub-problem using artificial slack variables with appropriate weights. It uses a slightly modified version of the Pantoja-Mayne update for the Hessian of the Lagrangian, variable dual scaling and an improved Armijo-type step size algorithm. Bounds on the variables are treated in a gradient-projection like fashion.

The cost function and constraint gradients are computed numerically by central difference approximations. The term $\partial d / \partial \mathbf{x}_k$ is the gradient of the minimum distance and it can only be computed numerically, since an analytic expression for $d(\mathbf{x}_k)$ is not available. The standard two-sided finite difference approximation can be used, where the ith component of the estimated gradient vector $\hat{\mathbf{g}}_k(\mathbf{x}_k)$ is given by:

$$[\hat{\mathbf{g}}_k]_i = \frac{d(\mathbf{x}_k + \delta \mathbf{e}_i) - d(\mathbf{x}_k - \delta \mathbf{e}_i)}{2\delta} \tag{10}$$

where δ is a small positive real number and $\mathbf{e}_i$ denotes a vector with a one in the ith component and zeros elsewhere. Notice that the computation of $\hat{\mathbf{g}}_k$ requires 6 evaluations of the minimum distance function for each step k, i.e. a total of $6N$ evaluations for each iteration. This is very expensive computationally.

A very important issue with NLP optimization is that any solution procedure may converge to a *locally* optimal solution, depending on the initial solution $\mathbf{X}_0$. In order to compute good quality initial trajectories and cover many – if not all – possible initial trajectories, a randomized kinodynamic planner is used. In the proposed two-phase approach, a random-tree based randomized planner is used to compute feasible control sequences $\hat{\mathbf{u}}$ of length N-1, which result in low cost collision-free sequence $\hat{\mathbf{x}}$ of length N. The best of these sequences is fed into the NLP solver.

IV. Decreasing cost randomized planning

The probabilistic planner used in this paper has been developed based on a biased bidirectional RRT path planner [4], which has been extended so that it computes a sequence of paths of monotonically decreasing cost. The standard bb-RRT algorithm grows simultaneously two tree data-structures, T_1, T_2. The initial state is inserted in the root-node of T_1 and the goal state in the root-node of T_2. In each iteration, each tree T_i is expanded towards a new state $\mathbf{x}'$. This state is either a *random* feasible state with probability p_1, or the root of T_j with a goal-bias probability p_2. To extend tree T_i, the bidirectional RRT finds the state $\mathbf{x}_{nn}^{Ti}$ which is closest to $\mathbf{x}'$ (nearest-neighbor), chooses a feasible control input $\mathbf{u}_{nn}^{Ti}$, which brings the new state $\mathbf{x}_{new}^{Ti}$ closest to $\mathbf{x}'$ and inserts it into a new node. In general, the new state is computed by integrating for a small time interval a state transition equation of the form $\mathbf{x}_{new}^{Ti} = f(\mathbf{x}_{nn}^{Ti}, \mathbf{u}_{nn}^{Ti})$ (forward integration if i=1, else backward). In the same iteration, the other tree T_j is expanded towards the new state $\mathbf{x}_{new}^{Ti}$, in an effort to connect the two trees, by computing a new state $\mathbf{x}_{new}^{Tj} = f(\mathbf{x}_{new}^{Ti}, \mathbf{u}_{nn}^{Tj})$ and inserting it in T_j. At the next iteration the indices i, j are swapped and the procedure is repeated until the two trees share a common state within some tolerance. The one-metric can be used, i.e., for each component k of the state vectors $\| \mathbf{x}_{new}^{T_1} - \mathbf{x}_{new}^{T_2} \|_k < \varepsilon$. Once the two trees connect, a collision-free sequence $\hat{\mathbf{x}}$ (and the corresponding $\hat{\mathbf{u}}$) is computed by constructing for each tree a sequence $\hat{\mathbf{x}}^{\mathrm{T}i}$ from the tree-root to the common state, and appending the two sequences $\hat{\mathbf{x}} = [\hat{\mathbf{x}}^{\mathrm{T}_1} \;\; \hat{\mathbf{x}}^{\mathrm{T}_2}]$. If the trees do not meet after $K_{\max}$ iterations, the algorithm terminates with an empty path.

The biased bidirectional RRT algorithm has been extended so that it computes paths of monotonically decreasing cost. Each tree node contains the state and control vectors $\mathbf{x}_k, \mathbf{u}_k$ and is associated with a cost function $\psi(\mathbf{x}_k, \mathbf{u}_k)$ which expresses some optimization criterion. The cost of a feasible path is the summation of the costs of its nodes. The extended planner does not terminate as soon as the first feasible solution has been reached. Instead, it continues sampling the state space and trying to connect the two trees, in order to come up with paths of lower-cost. Every time RRT finds a new feasible solution its cost is computed and this new solution is accepted only if its cost is lower than the best cost so far. Furthermore, all nodes in each tree for which the cost of the path that connects them to the corresponding tree root-node is greater than the new best path-cost can be deleted from the tree, since their further expansion cannot lead to paths with lower cost.

RRT is a probabilistically complete planner, i.e. a feasible path will be computed with probability which approaches one as the computation time grows to infinity.

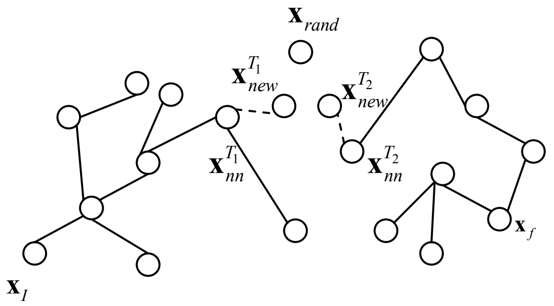

Fig. 2 Snapshot of trees' expansion

In practice, for most real problems a feasible path is reached in a finite number of iterations, which largely depends on the complexity of problem. The extended planner inherits this property. Furthermore, as its execution time grows longer, new lower-cost paths keep being discovered. However, the probabilistic convergence of the planner to a *globally optimal* path has not been formally established. We should also note that the solution of the planner is actually a trajectory, since each node carries time information.

VI. Simulation Experiments

The RRT-based planner and the direct transcription algorithm were implemented in C++. The NLP numerical procedure used was *donlp2-intv-dyn* by Prof. Spellucci [14], [15] and can be found at http://plato.la.asu.edu/donlp2.html. Collision detection and minimum distance computations were performed by the *PQP* Library [16], [17].

A number of simulation experiments were performed on a Pentium 2.6 GHz single-CPU system, to investigate the functionality and performance of the two-phase planner. The length of the vehicle's axis (front-to-rear wheels) was set to 2 meters. The steering angle constraint was set to $|\phi_{\max}| \leq 45^o$. The optimization criterion in all the cases was chosen to be the total distance D traveled by the vehicle. This can be shown to be

$$J = \sum_{k=0}^{N-1} |v_k| \Delta T = \sum_{k=0}^{N-1} (\mathbf{u}_k^T \mathbf{Q} \mathbf{u}_k)^{1/2}, \qquad (11)$$

where $\mathbf{Q} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ and ΔT is the discretization step, which in these simulations was fixed, and equal to 0.1 sec.

A. Kinematic Planning

The first set of experiments was conducted in the absence of obstacles and by using only the kinematic component of the vehicle's state equations (6), i.e., the first three equations. The vehicle's initial state was set to $\mathbf{x}_0 = [0\ 0\ \pi/2]$ and its goal state $\mathbf{x}_g = [1\ 0\ \pi/2]$. Thus, the vehicle was commanded to 1 m sideways, while facing north in both the start and goal positions. Figs. 3, 4, 5 and 6 show (with dotted lines) four paths of monotonically decreasing cost, as they were computed by RRT, which was executed with a maximum of 20000 nodes in each tree. The paths had 96, 68, 49 and 43 points, and required 45, 16, 1.1 and 1.5 minutes respectively. Since no obstacles were present to constrain the solution, the RRT produced paths of very different geometries. The solid lines show the corresponding paths as they were transformed after optimization by the NLP procedure applied on their entire length.

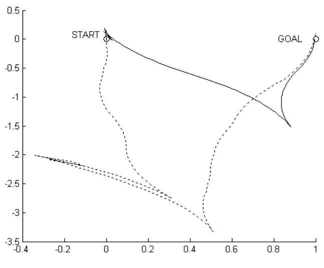

Fig. 3 RRT length: 9.49; optimal length: 4.07

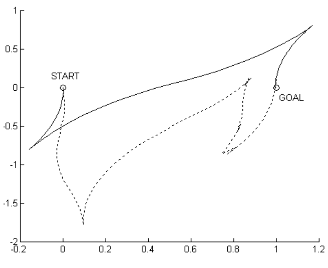

Fig. 4 RRT length: 6.72; optimal length: 3.82

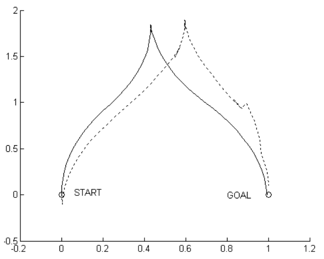

Fig. 5 RRT length: 4.76; optimal length: 4.02

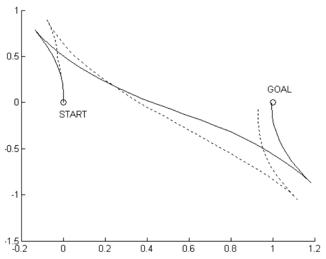

Fig. 6 RRT length: 4.20; optimal length: 3.83

Next, an optimal motion was computed among obstacles. The robot's initial state is $\mathbf{x}_0 = [0\ 6\ \pi/2]$ and its goal state is $\mathbf{x}_g = [2\ 10\ \pi/2]$. In Fig.7 the best path computed by the first phase is shown, together with the corresponding random tree. The path's length is 30.87 meters and its size 310 points. In Fig. 8 the optimal path is shown after the optimization.

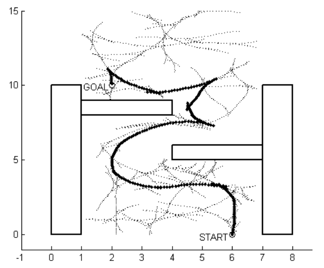

Fig. 7 Two-dimensional projection of a developed random tree and path

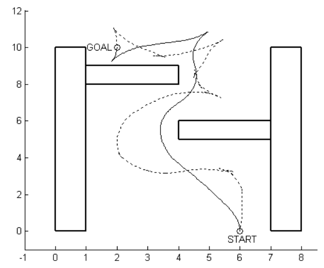

Fig. 8 RRT length: 30.87; optimal length: 16.9

In a third case study, the initial state of the vehicle was set again to $\mathbf{x}_0 = [0\ 0\ \pi/2]$, but this time the desired final position was $\mathbf{x}_g = [1\ 0\ 3\pi/2]$. The RRT planner produced a motion which was highly erratic (Fig. 9 dotted line). This was due to the fact that no dynamics were involved in its generation and the velocity could change and even reverse instantly (Fig.10). In this case the NLP optimization was not able to smooth the motion (Fig. 9 solid line) and got trapped into a local minimum, which was far from the optimal. Such undesirable situations can be avoided by the incorporation of dynamics in the RRT motion generation and the optimization.

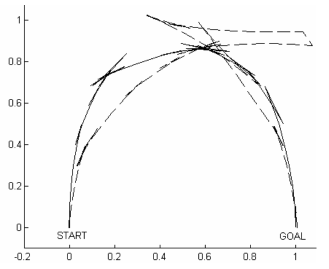

Fig. 9 RRT length: 7.28; optimal length: 6.28

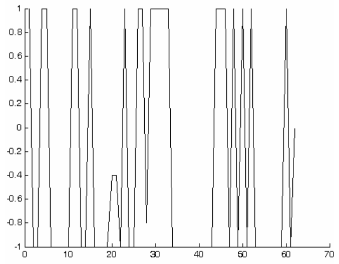
Fig. 10 RRT motion velocity profile

B. Kinodynamic Planning

In the second set of experiments the full dynamics of the vehicle were used. The maximum acceleration and deceleration was set to $|a| \leq 2$ km/h/s, the maximum forward velocity to 10 km/h, the maximum backwards velocity to -5 km/h, and the maximum wheel turning speed to $|\omega| \leq 45^o$ /s. The length of the vehicle's axis was 3m.

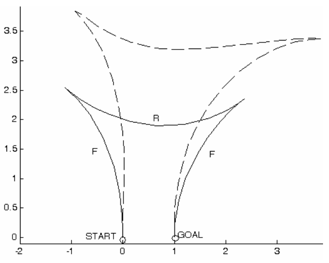

Fig. 11 RRT length: 13.99; optimal length: 9.42.

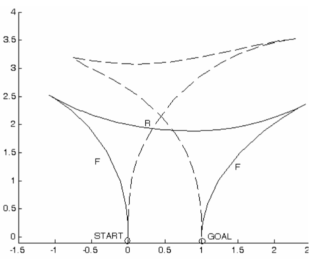

Fig. 12 RRT length: 11.57; optimal length: 9.44

Figures 11 and 12 depict the x-y coordinates of two paths computed by RRT (dotted lines) and the corresponding optimal paths. Forward motion is marked with (F) and reverse motion with (R). In Fig.12 RRT produces a "fish-tail" motion, well known in agriculture for turning at field headlands. However, the truly optimal motion turns out different. The velocity and steering angle profiles of the motions in Fig. 11 - before and after optimization - are shown in Fig. 13 and Fig. 14 respectively. Clearly, the motion after optimization is not only shorter but much smoother.

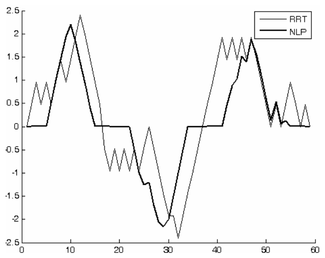

Fig. 13 Linear velocity profiles

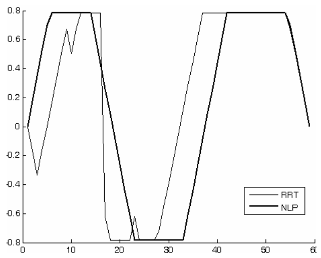

Fig. 14 Steering angle profiles

VII. Conclusions and Future Work

The objective of this work was to compute near-optimal motions for autonomous robotic vehicles. A two-phase approach was adopted, where an RRT-based randomized planner produces a sequence of motion trajectories of monotonically decreasing cost. In a second phase, the best motion computed by RRT is optimized, based on user-defined criteria. This problem was formulated as an optimal control problem and a direct transcription method, together with an NLP optimization algorithm were used to solve it numerically.

As a case study, near-optimal paths for a non-holonomic car-like robot were computed. The NLP optimization always converged to locally optimal solutions. In the case of kinematics planning the RRT may produce motions which are too erratic. In such cases the optimization may fail to improve the motion. However, this problem did not appear during the kinodynamic experiments, because the RRT motions were much smoother. In the presence of obstacles which restricted the shape of the possible paths, the randomized planners produced paths of similar "quality".

The computation times were big, in the order of a half hour, or more. The main reason for this is that the solution sequences computed by the RRT planner were long (N equals 40 to 300 points). The dimension of the NLP problem produced by the transcription method was $5N$ and $7N$ for the kinematic and kinodynamic planning cases respectively, i.e., it involves 200 to 2100 variables. The NLP procedure used was based on dense-matrix algebra, which means that the Jacobian and Hessian matrices computed – and inverted – were huge.

From these preliminary results it seems that two-phase path planners, which combine the search power of randomization with numerical optimization for path refinement, can offer a practical tool for the solution of motion planning problems. Additionally, direct transcription combined with NLP solvers can be a useful tool for the optimization of short paths (small number of points), or path-segments. For large paths though, especially when state constraints are present, its computational requirements become prohibitive and sparse-algebra optimization tools should be used.

References

[1] J. Barraquand and J.-C. Latombe, "A Monte-Carlo algorithm for path planning with many degrees of freedom", *Proc. IEEE Int. Conf. Robot. & Autom.*, pp. 1712-1717, 1990.

[2] S. Carpin and G. Pillonetto, "Robot motion planning using adaptive random walks, *Proc. IEEE Int. Conf. Robot. & Autom.*, pp. 3809-3814, 2003.

[3] E. Mazer, J. M. Ahuactzin, and P. Bessiere, "The Ariadne's clew algorithm", *Journal of Artificial Intelligence Research*, 9:295-316, 1998.

[4] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning, *Intl.l Journal of Robotics Research*, 20(5):378–400, May 2001.

[5] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces", *Intl. Journal of Computational Geometry and Applications*, 9(4-5):495–512, 1999.

[6] G. Song, S. L. Miller, and N. M. Amato, "Customizing PRM roadmaps at query time", *Proc. IEEE Int. Conf. Robot. & Autom.*, pp. 1500–1505, 2001

[7] J. Kim, R.A. Pearce and N.M. Amato, "Extracting Optimal Paths from Roadmaps for Motion Planning", *Proc. IEEE Int. Conf. Robot. & Autom.*, pp. 2424-2429, 2003.

[8] M. Yamamoto, M. Iwamura, and A. Mohri, "Quasi-timeoptimal motion planning of mobile platforms in the presence of obstacles", *Proc. IEEE Int. Conf. Robot. & Autom.*, pp. 2958—2963, 1999.

[9] J.M. Philips, N. Bedrossian and L.E. Kavraki, "Guided Expansive Spaces Trees: A Search Strategy for Motion- and Cost-Constrained State Spaces", *Proc. IEEE Int. Conf. Robot. & Autom.*, pp.3968-3973, 2004.

[10] D.E. Kirk. *Optimal Control Theory: An Introduction*, Prentice Hall, 1970.

[11] A.P. Sage and C.C. White, III, "Optimum Systems Control", 2nd ed., Prentice Hall, 1977.

[12] J. Betts, "Survey of Numerical Methods for Trajectory Optimization", *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 2, pp. 193-207, 1998.

[13] S.G. Vougioukas. "Optimization of Robot Paths Computed by Randomized Planners", IEEE Intl. Conf. on Robotics and Automation, pp. 2160-2165, April 2005, Barcelona, Spain.

[14] P. Spellucci. An SQP method for general nonlinear programs using only equality constrained subproblems. Math. Prog. 82, pp. 413–448, 1998.

[15] P. Spellucci. A new technique for inconsistent problems in the SQP method. Math. Meth. of Oper. Res. 47, pp. 355–400, 1998.

[16] S. Gottschalk, M. C. Lin and D. Manocha,, "OBB-Tree: A Hierarchical Structure for Rapid Interference Detection", Technical report TR96-013, Dept. of Computer Science, University of N. Carolina, Chapel Hill, 1996.

[17] E. Larsen, S. Gottschalk, M. Lin and D. Manocha, "Fast Proximity Queries with Swept Sphere Volumes", Technical report TR99-018, Dept. of Computer Science, UNC Chapel Hill, 1999.