Real Time Continuous Curvature Path Planner for an Autonomous Vehicle in an Urban Environment

David Knowles
Mentor: Richard Murray
California Institute of Technology

20 September 2006

Abstract

An real-time algorithm based on Ariadne's Clew and A* is developed to plan dynamically feasible trajectories appropriate to a vehicle traversing an urban environment in a legal manner. Simple precomputed clothoids are used to build a tree of possible maneuvers which is searched to minimize a cost function depending on path length, steering effort and lane discipline. The algorithm dynamically adapts its map sampling resolution to suit the difficulty of the problem. Complex situations where reversing is required are handled naturally. A proof-of-concept prototype real-time velocity planning algorithm is demonstrated. This functionality is proposed and recommended for use as part of Team Caltech's entry into the DARPA Urban Challenge 2007.

In November 2007 DARPA will hold the Urban Challenge race, in which qualifying autonomous vehicles will attempt to navigate a route through an emulated urban environment to specified checkpoints in a safe, legal manner. It is the aim of this project to develop Team Caltech's existing software onboard the 2005 DARPA Grand Challenge (DGC) entry, Alice, to be able to plan a dynamically feasible, traffic-law abidding trajectory. The existing path planning module is designed for a static, desert environment. The new planner will have to deal with moving obstacles whilst also changing its emphasis from minimising traversal time to performing safe, legal driving. This report describes the development of a spatial planning algorithm that could be used either to produce appropriate seeds for the optimization stage of the existing framework or as part of a newly proposed spatial-temporal planner framework.

The planning module developed for the Grand Challenge 2005 has two stages: stage 1 samples a speed limit map over a grid aligned to the Route Data Definition File (RDDF) corridor, and performs a forward-only graph search to find the fastest route; stage 2 employs the Non-Linear Programming optimization package SNOPT to minimize traversal time in a receding horizon framework. More details can be found in [1]. This project focuses on development of the stage 1 planner.

Various planning methodologies have been considered and two candidates are proposed. The first is an receding horizon time traversal optimization framework, based on the 2005 DGC software but with dynamic obstacle avoidance as incorporated by Martin Larsson [2]. Various problems are anticipated with this approach:

- time traversal minimisation is difficult to reconcile with safe, predictable driving
- how to incorporate legal constraints
- the receding distance horizon framework is does not allow for many urban situations such as queuing
- no method exists to add costs to actions such as crossing lanes
- the gradient descent method struggles with the binary obstacles found in urban environments
- unnecessarily complicated for highly constrained urban situations

There are also problems generating a useful seed for optimization. The rationale behind the existing split into two stages is that the stage 1 planner approximates a solution considering only spatial constraints. However, to generate a useful seed when dynamic obstacles are involved we must consider the coupling between the spatial and temporal problems: whether we need to change direction to avoid another vehicle depends upon the time we get there, which in turn depends on the spatial route we took. To generate the seed the full spatial-temporal problem must be solved, so the complexity is not reduced. Larsson [2]

adapted the optimization stage using the output from my spatial planner as a seed but noted that this can cause problems for convergence.

I propose an alternative planning architecture: divide the problem into spatial and temporal planning problems. A spatial planner generates a feasible spatial path up to a spatial horizon a given distance away, based on the static environment: legal constraints such as lane boundaries and static obstacles such as parked cars. A velocity planner maximizes the distance traveled along this spatial path within a given time horizon, based on the dynamic environment, i.e. the predicted movement of other vehicles being tracked. Each planner obeys the appropriate vehicle dynamics: the spatial planner minimum turning radius, the velocity planner maximum acceleration and maximum speed for the road and for the turning radius.

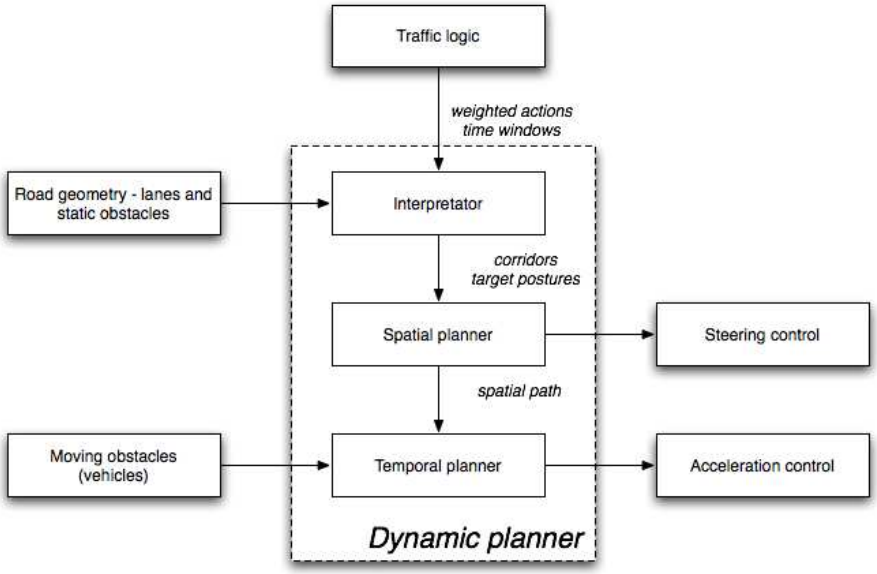

Figure 1: Proposed planning architechure. Clean linear data flow is achieved: static and dynamic maps are considered separately; spatial and velocity planners output to steering and acceleration controllers respectively.

The advantages are that this approach

- has greatly reduced complexity at both stages which is beneficial for development, computational load and reliability
- has very clean data flow as shown in Figure
- allows for a trajectory where we come to a stop until the end of the time horizon, e.g. behind a queued car

- is analogous to how humans drive: in most cases we decide where we want to go and then drive that path at a safe speed given other vehicles [3]

Since the choice between these methods is still under high-level discussion, the spatial planner developed is designed to integrate into either framework. It could be either to produce more useful, realistic seeds for the optimization stage or as part of the spatial-velocity planner framework.

The first cut of the spatial planner was an A* based algorithm [4]. The legal zone was sampled into a grid, in much the same way as the existing stage 1 planner, but without restricting movement to the direction of the corridor. A grid-based A* algorithm does not take vehicle dynamics into account. To account for this I proposed calculating the change in yaw angle from one node to the next and adding a weighted multiple to the cost function G for the edge. This method would also allow the turning radius to be constrained, albeit in a approximate manner. A major difficulty encountered was how to smooth the trajectory and remove artifacts resulting from the grid-based sampling of the map.

Because of the inherent problems of imposing a grid based graph search onto the spatial planning problem, it was decided to move to a methodology using a tree of simple, feasible curves to describe potential trajectories [3]. Clothoid curves were chosen because they allow a continuous curvature trajectory to be specified. A clothoid has curvature which varies linearly with arc length, and is therefore equivalent to turning the steering wheel at constant speed. When the maximum curvature is reached, i.e. the minimum turning radius, the curve becomes the arc of a circle [5]. The method used is based on Ariadne's Clew [3], a general planning technique consisting of two complementary algorithms, *search* and *explore*. *explore* iteratively grows a tree of states (positions and orientations) reachable by simple maneuvers. *search* tries to find a simple trajectory from any existing node of the tree to the target waypoint. *explore* effectively discretizes the map but using dynamically feasible continuous curvature maneuvers rather than an artificial grid. This method extends naturally to complex maneuvers such as parking or K-turns by adding reverse maneuvers to the set that the sub-algorithms consider.

In [3] *explore* is specified as an optimization problem itself: to maximize the distance of a new node in the tree from any existing node. This is a large parameter space: the new node can be at the end of a clothoid curve with any length, any curvature and originating at any existing node in the tree. This is one of the main reasons Scheuer's work involved planning times of around four minutes, orders of magnitude too slow for real-time planning. I simplified this problem in two ways:

1. precomputed simple curves for a predefined range of length and clothoid sharpness as shown in
2. used the concept of a heuristic from A* to direct *explore* towards the target point rather than growing the tree isotropically

An example of a tree grown by explore is shown in Figure

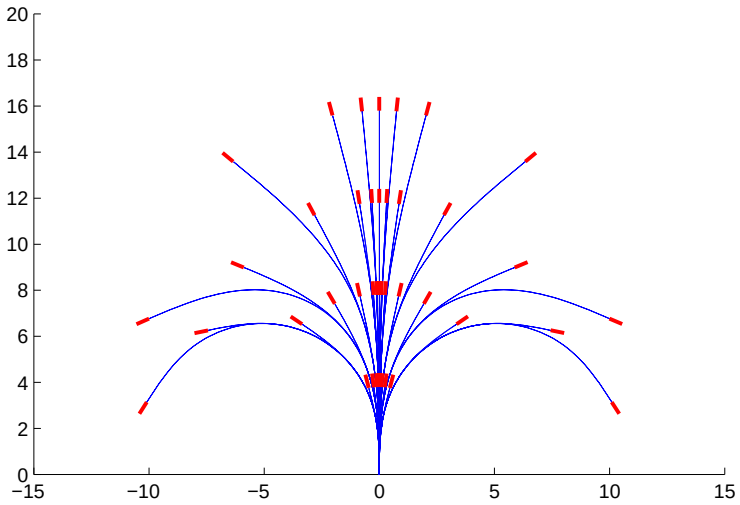

Figure 2: The set of precomputed elementary paths used by *explore*.

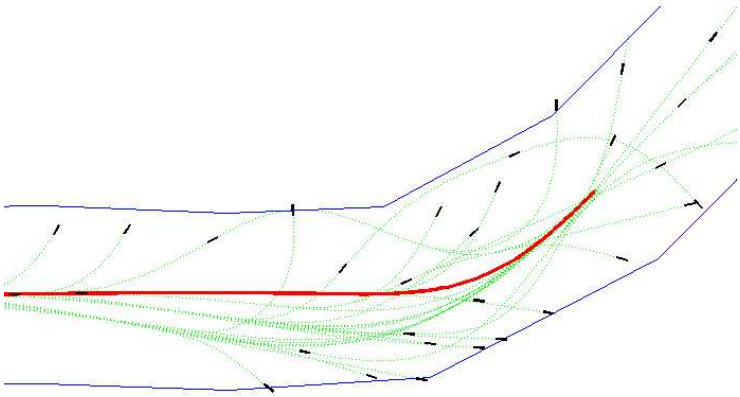

Figure 3: An example of node placement by *explore*. The black dashes are nodes, the green lines the edges between them, and the red line is the actual trajectory chosen. Note that for clarity a short run time (0.02s) was used to generate less nodes.

Initial development of the algorithm was done in open loop: a single path was generated for a given situation and was viewed in Matlab to ensure it was reasonable. A run time of 0.1 seconds is in line with our expectations for the refresh rate of the planner and gives good solutions. Some of the results are shown in Figure .

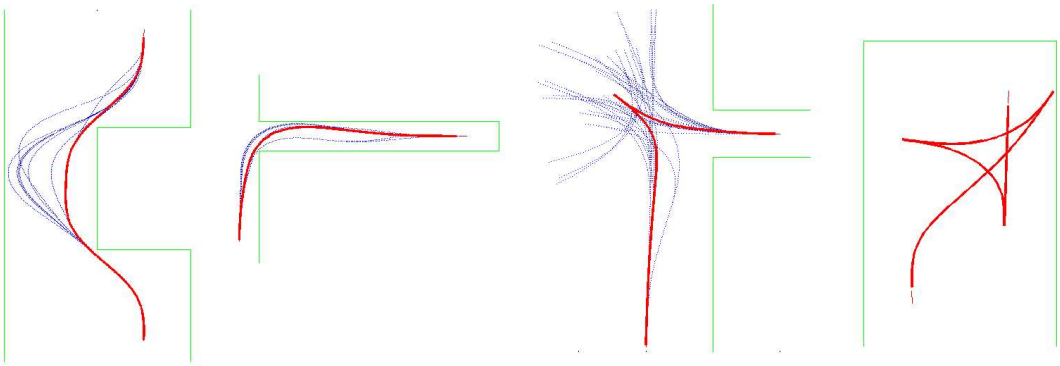

Figure 4: Example unit tests of the algorithm. Green is the legal driving zone, blue are feasible paths and red is the path chosen to minimize the cost function. From the right these show: passing a parked car, turning right into a narrow lane, reverse parking, a K-turn. Note that the right two simulations were run with reversing allowed, and that these solutions come naturally from the algorithm.

For closed loop, continuous, real-time planning the 2005 DGC planner replanned from scratch at every iteration. This is computationally wasteful and can result in oscillations between local minima in the cost function on successive planning cycles. As a more robust solution I chose to continually adapt the graph as improved data from sensing is obtained. This requires several adaptations to the open loop algorithm:

- once the vehicle has set off towards the first planned node the tree must be "pruned" to remove nodes that would have been reached by any of the other choices of initial node.

- as the map is updated it may be discovered that paths thought to be legal are not so, and these must be deleted

- the cost of certain edges between nodes may change, for example if road-finding misidentified a lane boundary, in which case the cost for all nodes downstream must be updated

- the target waypoint may move (especially in a receding horizon framework), requiring the *search* algorithm to be rerun and the heuristic, H, to be recalculated for all nodes

These developments were tested in simulation using the *simulator*, *trajFollower* and *GUI* modules from the DGC 2005 software. Careful algorithm design has meant that the planner actually completes these operations at faster than 10Hz, our original specification.

Testing has been undertaken on Alice, Team Caltech's autonomous vehicle. Two distinct testing frameworks were used: in the first, the output from the spatial planner was used as a seed for the optimization planner; in the second, it was the actual trajectory passed to *trajFollower*. Note that in the second case manual throttle was used in the absence of a velocity planning algorithm.

The closed loop results on Alice showed convergence problems where the planner would make significant changes to the upcoming trajectory which were not seen in simulation. These may however have been an artifact of feeding the planner with target positions that were too close to the vehicle. This result raised the suggestion of reincorporating a solution to the convergence problem used in DGC 2005: to associate a cost to the distance of each point on the new trajectory from the nearest point on the previous trajectory. I hope this will be unnecessary since the sampling of the map should ensure that a good approximation to the global minimum is always found, preventing the algorithm "getting stuck" in a local minimum.

While the main focus of this project has been on the spatial planning problem, I have also considered how the velocity planning problem might be approached if it was decided not to use the optimization based planner. Because the spatial part of the problem is solved by this stage, the velocity planner must consider the plane formed by s, the arc-length along the spatial path from our current position, and time, t. This is a still a difficult non-linear programming problem with non-holonomic constraints such as maximum acceleration/deceleration.

As a proof of concept prototype I designed an algorithm which optimizes over just one parameter: current spatial derivative of velocity, $k = \frac{dv}{ds}$. Using k rather than acceleration a greatly simplifies the problem so that closed analytic expressions can be found for the vehicle's projected speed and the time at which points along the trajectory will be reached. This allows the algorithm to access the *CDynMap* interface with vehicle tracking written by Martin Larsson. By querying this "dynamic map", which contains information on all the vehicles being tracked, it is possible to determine whether it is feasible for the vehicle to be at a particular space-time coordinate in the future.

We have discussed the interface between the traffic and path planning modules: our recommendation is that traffic planning passes path planning a set of legal driving zones, defined as polygons, and a target waypoint in each zone. I feel that vehicle avoidance should not be done at two levels, both in the proposed traffic planning and path planning modules. Potentially the most coherent method would be to have a dual layer dynamic map - one layer containing the physical obstacles that we must avoid, the other containing obstacles representing the region around other vehicles that we should not enter if possible.

I have developed an intelligent spatial planner appropriate to use in an urban driving environment that could be used as a seed for a second optimization stage

operating in space and time, or just time. There is still significant work to be completed within the navigation architecture if Alice is to compete successfully in DUC 2007.

- Decide on and develop a stage 2 approach: traversal time minimization, pure velocity planning, or other

  Interfaces: with road-finding using *CVectorMap* vectorized map frameworks, with traffic planning using legal drives zones, with tracking and prediction, with supervisory control and with trajFollower (reversing must be incorporated)

- Anisotropic growth of other vehicles in *CDynMap* to take into account Alice's orientation at each point along the spatial path

- Test the algorithm's behavior in complex driving situations using the newly developed traffic simulator

1 Methods

The following is a detailed technical description of the functionality of the spatial planner.

1.1 Spatial planner

I have designed a spatial path planning algorithm which explores a tree of elementary paths as defined in [5] and [6]. An elementary path is defined as a curve with symmetric linear variation of curvature, κ with arc length, s:

$$\kappa(s) \quad = \quad \begin{cases} \min(\sigma s, \kappa_{max}) & 0 \leq s < \frac{l}{2} \\ \min(\sigma(l-s), \kappa_{max}) & \frac{l}{2} \leq s < l \end{cases} \tag{1}$$

where σ is known as the clothoid sharpness. The angle, θ, along the path is found by integrating the curvature:

$$\theta(s) \quad = \quad \int_0^s \kappa(s)\, ds \tag{2}$$

Finally the (x, y) coordinate is found as follows:

$$x(s) \quad = \quad \int_0^s \cos\theta(s)\, ds, \qquad\qquad y(s) \quad = \quad \int_0^s \sin\theta(s)\, ds \tag{3}$$

Both (2) and (3) are currently computed using trapezoidal numerical integration. More advanced approximations are unnecessary as it is not a requirement for TrajFollower that the curvature is exactly linear. The planner precomputes the elementary paths it will use, which have a finite range of sharpness, σ, and arc length, l. The elementary paths therefore consist of straight lines, clothoids (curves whose curvature varies linearly with path length) and arcs of circles. The set of elementary paths currently implemented was shown in Figure .

The algorithm I have developed is based on Ariadne's Clew [3] but with key features of A*, specifically a cost function and a heuristic, an estimate of the cost from the current landmark to the final landmark. We refer to nodes of the tree as landmarks or postures to emphasize the fact that they include orientation as well as position. The path cost function is calculated as the path length plus a factor to penalize steering effort:

$$G_p(l, \sigma(s)) \quad = \quad \int_0^l ds + A \int_0^l \sigma(s)\, ds \tag{4}$$

At each node a cost is added associated with the proximity to and alignment with the nearest lane boundary.

$$G_n \quad = \quad \frac{1}{max(1.0, d^2)} + sin^2(\Delta\theta) \tag{5}$$

where d is the distance to the nearest lane boundary and $\Delta\theta$ is the difference in yaw. The total cost is then calculated as $G = G_p + l \cdot G_n$.

Ariadne's Clew uses two complementary algorithms, *explore* and *search*. *explore* generates a new landmark by traversing an elementary path from an existing landmark. It solves an optimization problem: maximising the minimum distance from the new landmark to any of the existing ones. Every time *explore* generates a new landmark, *search* checks whether a simple trajectory (see Section 1.2) can be found from the new landmark to the target landmark, q_t. More details can be found in [3]. The algorithm I have developed works as follows, where Λ_n denotes the set of most recently generated landmarks.

1. Add the start posture, q_s to the tree, Λ, analogous to A*'s open list. $\Lambda_n = \{q_s\}$
2. If *search* finds a solution between Λ_n and q_t, then calculate the complete path and add it to the set of feasible paths.
3. If time is up, Go to 8.
4. Find the landmark, λ_l, which has the lowest F value, which has not yet been explored.
5. $\Lambda_n = explore(\lambda_l)$ and mark λ_l as having been explored.
6. *For each* λ in Λ_n calculate the cost, G from q_s to λ, the heuristic, H from λ to q_t, and the total, $F = G + H$.
7. Go to 2
8. Return the feasible path with the lowest cost.

I will now explain the implementation of the *search* and *explore* algorithms in more detail.

1.2 Search

The *search* algorithm is based on Kanayama's method [7] but adapted for clothoids rather than cubic spirals using results from [6]. *search* looks for a simple path between from landmark p_1 to landmark p_2. We are only able to join two landmarks using an elementary path as defined by (1) is they are *symmetric*. Two landmarks are said to be symmetric if and only if:

$$\theta_1 - \beta = -(\theta_2 - \beta) \tag{6}$$

where θ_1, $theta_2$ and β are defined by Figure 1.2 and by:

$$\tan\beta = \frac{y_2 - y_1}{x_2 - x_1} \tag{7}$$

If the landmarks are not symmetric then we try to connect them using a bi-

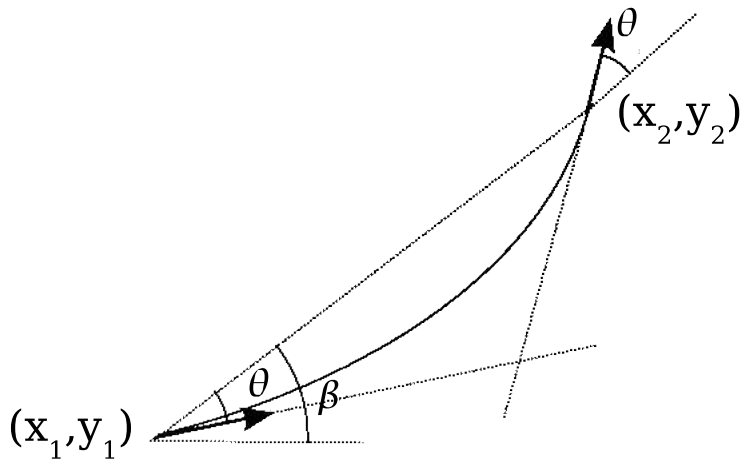

Figure 5: Two symmetric postures and a connecting elementary path.

;

elementary path, i.e. the union of two elementary paths and an intermediate landmark. Firstly we try to find this intermediate posture, q which is symmetric to both p_1 and p_2. If the landmarks are parallel, i.e. the deflection, $2\alpha = \theta_2 - \theta_1 = 0$, then this is straightforward. Kanayama [7] showed that the intermediate posture lies on the line through p_1 and p_2, with the total curvature of the path minimized at the midpoint.

For non-parallel postures, Kanayama [7] proved that the locus of q is a circle, center, p_c such that

$$p_c = \begin{pmatrix} x_c \\ y_c \end{pmatrix} = \frac{1}{2}\begin{pmatrix} x_1 + x_2 + c(y_1 - y_2) \\ y_1 + y_2 + c(x_2 - x_1) \end{pmatrix} \tag{8}$$

where $c = \cot\alpha$. We choose the actual position of q on this circle by calculating the cost of the path for different positions on the circle (separated by constant arc length) and choosing the lowest.

Once the position of q is determined its orientation is determined from (6). The parameters of the two elementary paths connecting p_1 to q and q to p_2 respectively are determined by:

$$\sigma = 4\pi \operatorname{sgn}(\alpha) \frac{D_1(|\alpha|)^2}{r^2} \quad (9)$$

$$l = 2\sqrt{\frac{2\alpha}{\sigma}} \quad (10)$$

where r is the distance from p_1 to p_2 and D_1 is defined over $[0, \pi]$.

$$D_1(\alpha) = \cos\alpha F_C\left(\sqrt{\frac{2\alpha}{\pi}}\right) + \sin\alpha F_S(\sqrt{\frac{2\alpha}{\pi}}) \quad (11)$$

$$F_C(x) = \int_0^x \cos(\frac{\pi}{2}u^2)\,du \quad (12)$$

$$F_S(x) = \int_0^x \sin(\frac{\pi}{2}u^2)\,du \quad (13)$$

F_C and F_S are the Fresnel Cosine and Sine integrals, which I use a power series expansion to calculate. Once these parameters have been found and check for feasibility (maximum curvature and maximum sharpness) Equations (1-3) can be used to generate the paths.

1.3 Explore

In [3] a genetic optimization algorithm tries to maximize the minimum distance of the new landmark from any existing landmark. The advantage of this approach is that it adapts to the difficulty of the planning problem. A simple situation such as going down a lane with no obstacles is solved using few landmarks, because *search* rapidly finds complete feasible paths. However, a complex maze-like situation requires more landmarks, effectively giving a greater sampling resolution of the map.

Because my algorithm has to run in real-time I use a simpler explore function at the cost of completeness. Firstly, rather than searching over all possible nodes we use a concept from A*. For each node we calculate the traversal cost from the start node, G, and an estimate of the traversal cost to the finish point, known as the heuristic, H. $F = G + H$ and the node with the lowest value of F is explored first, because it is the most promising.

Secondly, instead of allowing a curve to have any clothoid sharpness and length and thus requiring on-the-fly construction of these curves, a discrete range of these parameters are permitted allowing curves corresponding to all combinations of sharpness and length to be precomputed. The implementation uses the set of elementary paths shown in . These are rotated and translated to fit onto the landmark being explored. Paths which leave the legal driving region or pass through obstacles are deleted. A single parameters controls the resolution of the nodes in terms of the minimum distance that a new node is

allowed to be placed from any other existing nodes. This value is reduced only if no more nodes can be placed as a result. Testing showed that this was a useful technique to ensure the possible configuration spaced is filled completely before sampling the map at greater resolution.

Both *search* and *explore* can operate with reversing allowed. This means that from each landmark the algorithm can choose to go into reverse if the "reversing allowed" flag is set to true.

1.4 DGC 2005 Stage 1 planner

It is worth briefly explaining the operation of the DGC 2005 stage 1 planner for comparison. This simple algorithm undertakes the following steps:

- Generate initial seed from RDDF trackline
- Construct a grid of nodes aligned to trackline (excluding two circles on either side of vehicle we are unable to enter due to the steering angle constraint)
- Sample speed map at midway points between nodes and calculates traversal cost
- Graph search for cheapest route with restriction of forward progress
- Outputs this trajectory to the refinement stage optimization planner to be used as its seed

1.5 Prototype velocity planner

The velocity planner tries to maximise $k = \frac{dv}{ds}$. Acceleration can be expressed in terms of k:

$$\begin{aligned} a &= \frac{dv}{dt} && (14)\\ &= \frac{dv}{ds} \cdot \frac{ds}{dt} && (15)\\ &= k \cdot v && (16) \end{aligned}$$

Defining $s = 0$ at our current position and integrating gives

$$\begin{aligned} v &= v_0 + k \cdot s && (17)\\ \Rightarrow \frac{ds}{v_0 + k \cdot s} &= dt && (18) \end{aligned}$$

Integrating again we find a closed expression for time which can be evaluated and used as an input to *CDynMap*.

$$t - t_0 = \frac{1}{k} ln|1 + \frac{k}{v_0} \cdot s| \quad (19)$$

References

[1] Dmitriy L. Kogan and Richard M. Murray. Realtime path planning via nonlinear optimization methods. Master's thesis, California Institute of Technology, 2005.

[2] Martin Larsson and Richard Murray. Optimization-based refinement stage planner for an autonomous vehicle. Technical report, California Institute of Technology.

[3] A. Scheuer and Th. Fraichard. Planning continuous-curvature paths for car-like robots. volume 3, pages 1304–1311, 1996.

[4] Patrick Lester. A* pathfinding for beginners, 2005.

[5] A. Scheuer and Th. Fraichard. Continuous-curvature path planning for multiple car-like vehicles. pages 8–12, 1997.

[6] Alexis Scheuer. *Planification de Chemins A Courbure Continue pour Robot Mobile Non-holome*. PhD thesis, 1998.

[7] Y. Kanayama and B. I. Hartman. Smooth local path planning for autonomous vehicles. volume 3, pages 1265–1270, 1989.

Acknowledgments

I would like to thank my mentor Richard Murray for his enthusiasm and patience devoted to Team Caltech. Anyone with that much on their plate who is prepared to sit down with you to find that last Seg Fault is doing a good job! I thank Martin for fighting the monstrous optimization based planner despite my constant attack on its usefulness in for urban driving (I think he agreed eventually). I thank Noel duToit and Joel Burdick for their navigation team guidance (and pizza). Last but not least I thank Albert and Ken for looking after Alice for us and keeping us and our code branches safe on the field tests.