A Study of Polynomial Curvature Clothoid Paths for Motion Planning for Car-like Robots

Mihail Pivtoraiko, Alonzo Kelly

CMU-RI-TR-05-nn

The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

December, 2004

Abstract

In this report we present an overview of our investigation into constructing motion templates. We start with a successful implementation of elementary concepts that worked well on previous projects, then proceed with a generalization of that approach through a study of spatial distinctness of the same type of paths, and finally present a further generalization that looks at all possible motions through space.

Table of Contents

3. Introduction

Here we present some important results of our study in application of polynomial curvature clothoid paths as applied to constructing motion templates. A simple application of these concepts was done in the Simple Robot Simulator (SRS) as well as on other programs which used technology developed in SRS.

In this presentation we mainly build on that work by introducing several key improvements that make motion planning more efficient. We also discuss the results of a detailed investigation to improve the quality of planning, for example with respect to resolution completeness and other important qualities.

4. SRS Proof of Concept

The Simple Robot Simulator (SRS) featured a successful implementation of a *lattice*. It was an explicit 3D memory array[1], each cell of which was a graph node connected with other nodes with edges. Only the edges that corresponded to feasible motions were kept in the graph. This determination was based on the results of the Inverse Solution [4]. The algorithm for the creation of the lattice was as follows:

Algorithm 1

for each x_{goal}
  for each y_{goal}
    for each θ_{goal} {
      for each θ_{init} {
        if path $\tau : (0, 0, \theta_{init}) \rightarrow (x_{goal}, y_{goal}, \theta_{goal})$ exists and $\kappa_{max_\tau} \leq \kappa_{max}$ then
          create an edge between $(0, 0, \theta_{init})$ and $(x_{goal}, y_{goal}, \theta_{goal})$.
          create an edge between $(x_{offset}, y_{offset}, \theta_{init})$ and $(x_{goal} + x_{offset}, y_{goal} + y_{offset}, \theta_{goal})$ for any (x_{offset}, y_{offset}) such that $x_{goal} + x_{offset} \leq L_{lattice}$ and $y_{goal} + y_{offset} \leq L_{lattice}$.

In this manner, all the edges between all the nodes were represented explicitly. Due to extensive memory requirements, this algorithm could represent only a limited portion of C-space: the size of the lattice was 25 by 25 ($L_{lattice}$ =25), with 16 equal headings per node. The maximum outdegree of the resulting search graph was 33.

Yet, the algorithm worked very well for getting the vehicle out of tight cul-de-sacs in natural cluttered environments (see Fig. 1). Due to the limited size of the lattice, it was only able to make local motion plans.

The drawbacks of that approach were memory requirements that required that plans were limited to the immediate proximity of the vehicle. Of course, since all of the lattice was resident in memory, the computation was fast.

1. Actually implemented as a 1D array for efficiency purposes.

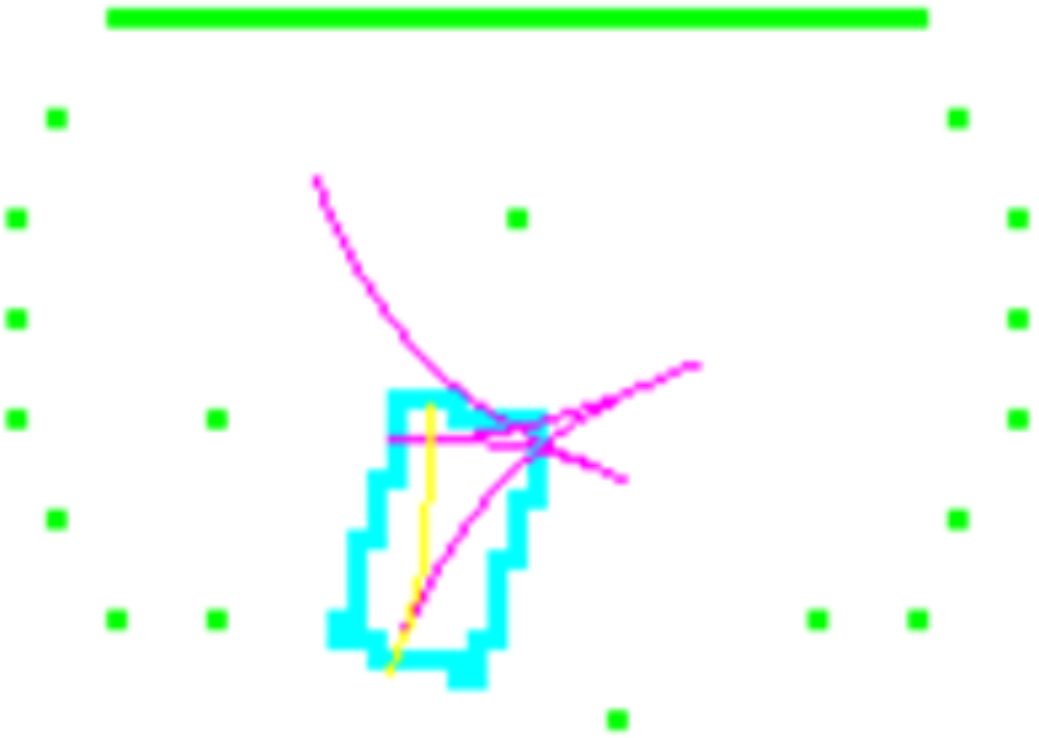

Figure 1: Example Lattice Plan. A motion plan that gets the vehicle out of the cul-de-sac has been automatically generated.

5. Implicit Lattice

As an effort to overcome the memory limitation an idea was suggested that there was no real need to copy the template everywhere in the lattice explicitly. Such a template could be assumed by the searching algorithm to exist at any point in the lattice. Two benefits of this representation were:

- only the small template is stored in memory
- it becomes possible to search the infinite C-space

The first guess was to use the same template as was used in SRS. There is little doubt that such an approach would work just as good as the one with the explicit lattice.

However, the size of the template used in SRS was completely arbitrary. It was largely dictated by the memory requirements for explicit copying of this template in the lattice. Moreover, we felt that there may be some limit to the required size of the template, so that

- there would be some practical explanation for the size of the template
- a template would be as large as is required to represent many more possible maneuvers than the previous limited template could allow.

It was suspected that there must me some certain maximum size of the template, *template horizon* L_H, which can allow generation of all possible motions through workspace. This was based on an intuitive argument that a path that connects initial and goal nodes, arbitrarily far apart, cannot go

infinitely long without hitting a node on the way. When it does hit some third node, it can be decomposed into sub-paths. It is then only necessary to keep only the two sub-paths in the template. They can be concatenated in various ways to re-create the original path, as well as infinitely many other paths. With this in mind, we started looking for the size of the template that would contain all elementary paths that could recreate any motion through workspace.

6. Minimal Set of Spatially Distinct Feasible Motions Using Inverse Solution

In this work we considered templates of various sizes in an effort to find such a lattice size so that all of its paths could be decomposed. Thus, we were primarily interested in decomposing the paths that ended on the perimeter of the template. If there are paths on the perimeter (by construction they are the longest paths in the template) that cannot be decomposed, then L_H is certainly beyond the size of this template.

A principled method for characterizing template sizes became necessary. We defined two terms:

- *template radius* -- it is simply the width and height of the template (both equal), R_T
- *angle radius* -- a measure of the number of discrete headings in the template, R_A; to define it, let us first look at the peculiarities of heading discretization.

6.1 Heading Discretization

In the process of discretizing continuous poses, it will be quite beneficial to use special angles for heading angles. If the goal is to get from the origin to a cell that is away three cells and up two (see the figure below), then it is important for the system to produce the result that can get the robot there on a straight path. Therefore, the angles of discretization should be $atan\left(\frac{n\Delta}{m\Delta}\right)$, where Δ is spacing between cells in y and x direction, and for symmetry is desired to be equal, and n and m are coordinates of a cell in discretized space. Hence, we define the notion of *angle radius*: it is the maximum value of n and m above. For a given angle radius there is a certain number of discretization angles (Fig. 2).

It is worth noting that this approach to heading discretization has some interesting properties. In particular, the number of discrete headings grows exponentially with angle radius (see Fig. 3).

When $R_A = 2$ we have only one angle between 0 and 45 degrees (26.6 deg.); when $R_A = 3$ there are 3 angles. We can predict that with increasing R_T the value of discretized headings will also decrease quite quickly. Indeed, as we see in Fig. 4 that shows the value of *a half of the maximum heading interval* as a function of angle radius.

As we see here, after the angle radius increases over 5, the discretized heading values decrease quite slowly. This is one of the observations that prompted us to make a conclusion early on that whereas increasing template radius was clearly beneficial for path elimination (see Section 6.2), the increase in angle radius *was not*. Almost no improvement can happen since discretized values change so little.

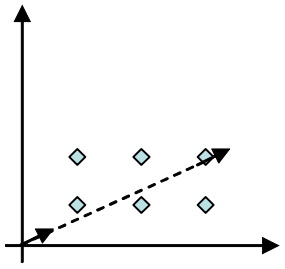

Figure 2: One of the useful headings. It is important for a n algorithm to be able to generate straight paths whenever possible

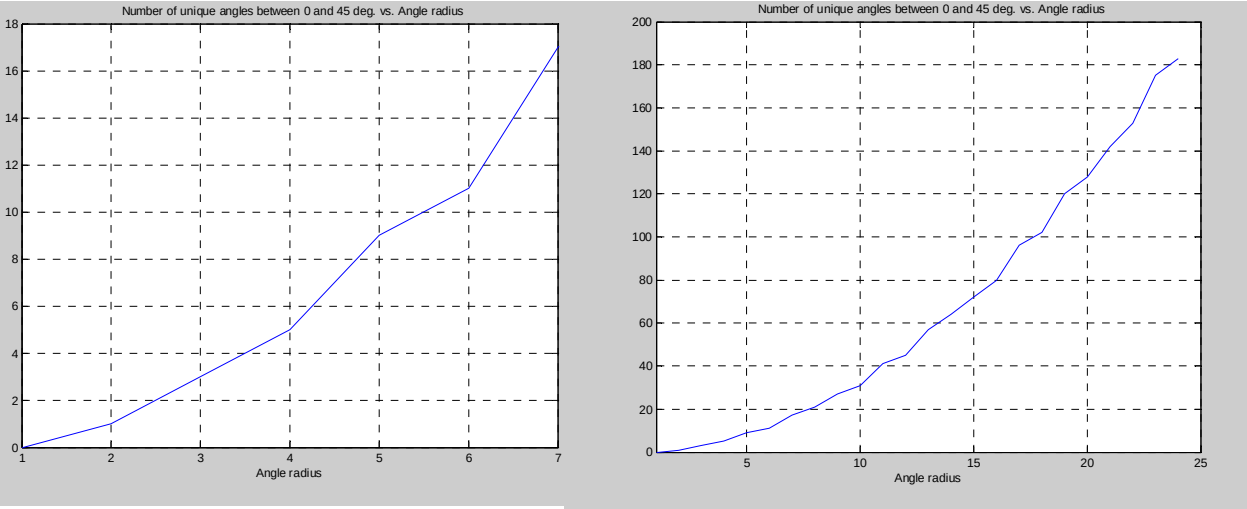

Figure 3: Number of unique angles vs. Angle Radius.

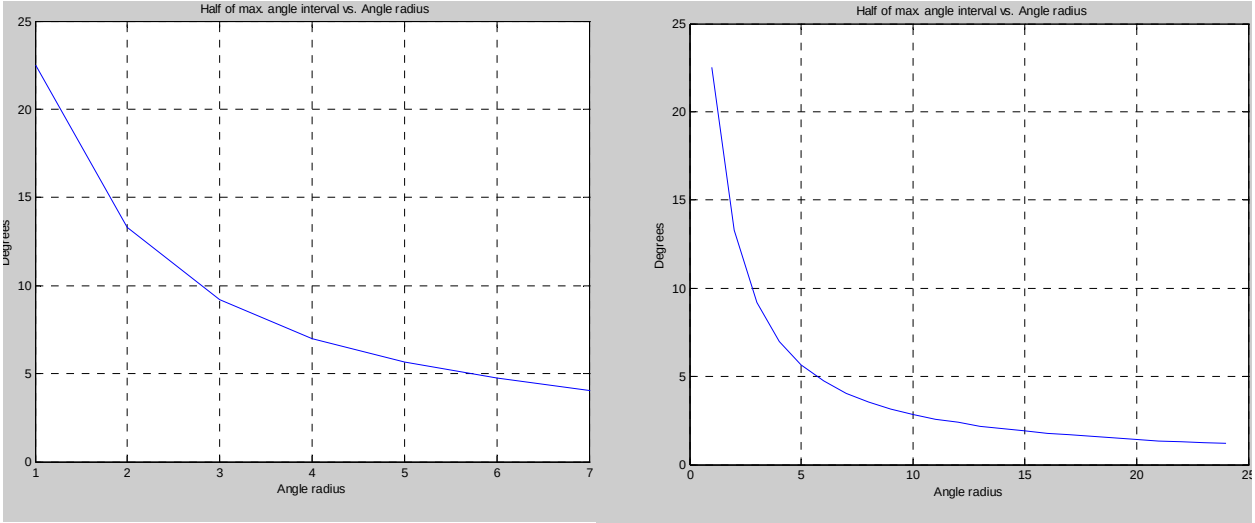

Figure 4: Discretized heading values vs. Angle Radius.

Another observation in support of this conclusion is that the number of headings, hence the total number of nodes in the template grows dramatically with the increase in angle radius. The added computational difficulty is not worth the gain.

In the case of uniformly spaced headings (as was done in SRS) the same arguments apply, as the increase of the number of headings is also exponential (think of 2, 4, 8, 16, etc.) and the discretized values that this process produces behave in the same way as demonstrated in Fig. 4.

6.2 Analysis of Path Proximity to Nodes

Once we reached some conclusions about the convenient discretization of the C-space, we wanted to verify our intuition that the longer the paths are, the more likely they are to cross other nodes in the template. We employed the following algorithm.

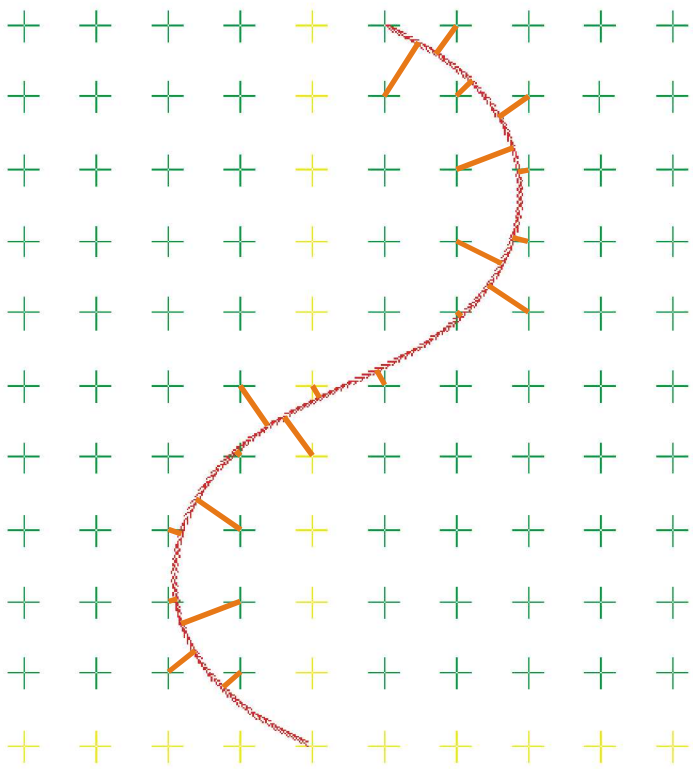

Figure 5: Illustration of the algorithm for calculating the path's minimum distance to any node. Orange highlights denote Euclidean distances to the nodes along the path. The minimum such distance is sought.

Algorithm 2

```
For each path in the template
   First, find the closest node, x_node, record its coordinates and distance ρ(x_path, x_node)
      Move ds along the path, calculate the new ρ(x_path, x_node) .
         If it is smaller, continue iterating
         Else, we are departing this node, use previous ρ as the minimum distance to
x_node
      Repeat move ds
   Repeat for other nodes along the path
   Keep a minimum of such distance to any node, ρ_min
```

For $\rho(x_1, x_2)$ the L_2-norm was used (Euclidean distance). This algorithm is illustrated in the Fig. 5. To understand the utility of this algorithm, consider a path whose minimum distance to any node is 0. This means that the path is perfectly aligned to a node along its way. In this case it is reasonable to eliminate this path from the lattice because it is composed from two sub-paths, that must already exist in the lattice because they are smaller than the original path and by construction the template includes all paths that start and end on the templates nodes and are within the template radius.

With these principles, a comprehensive study of such minimal distances to any node has been undertaken for paths ending on template *perimeter*. The results for a template of angle radius 2 are summarized in the graph in Fig. 6.

In particular, we see that for lattice radii of over 10 (quite small!) all paths come to within 20% of template node spacing (Δ). In other words, if we set the node tolerance to 20% (i.e. the path "hits

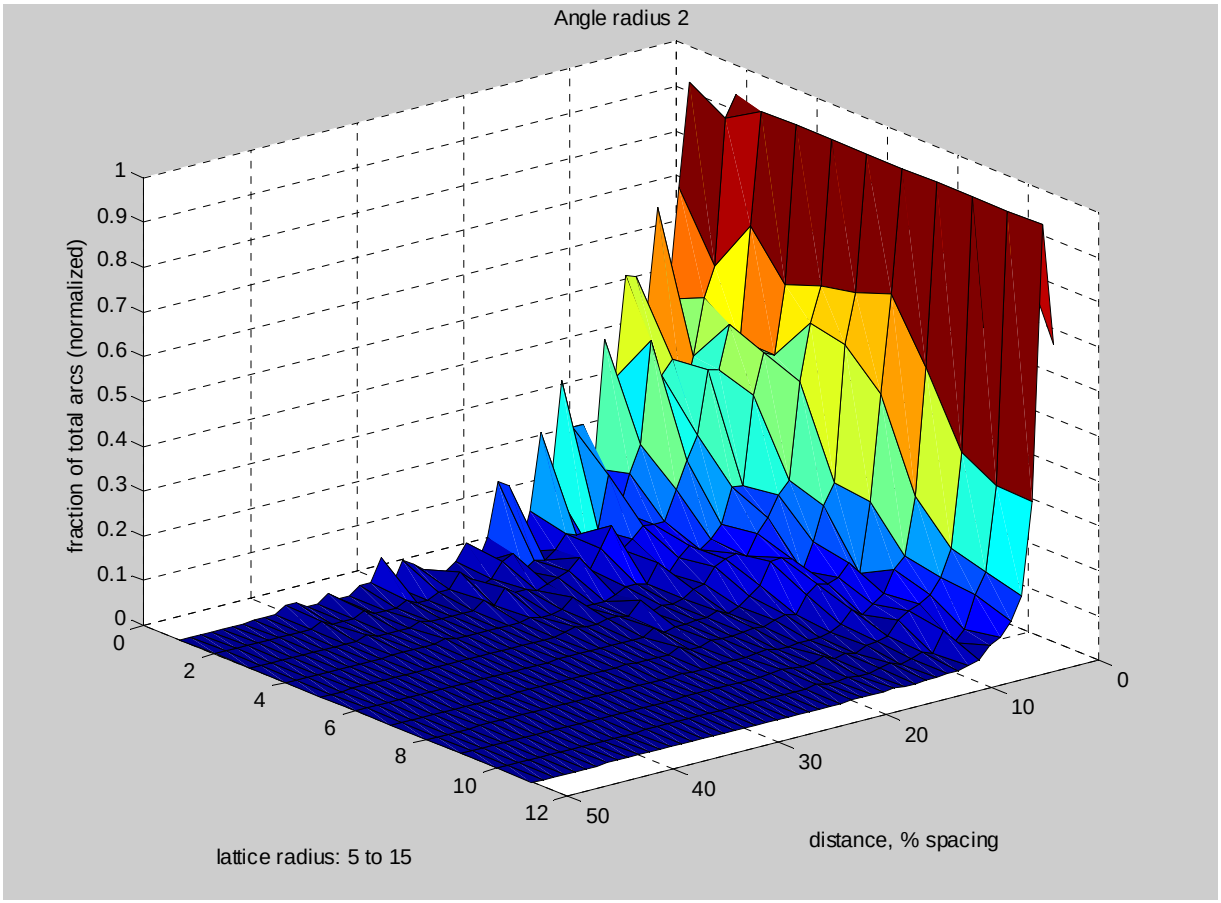

Figure 6: Summary of the minimum distance to any node study. This graph shows the percentage of the paths that have a particular value of minimum distance to any node, expressed as a percentage of 2D spacing between template nodes, given a particular template radius (here same as *lattice radius*).

the node" by approaching it inside this tolerance), then all of the perimeter paths in the template can be eliminated. This is a powerful conclusion. It means that if we adopt the above tolerance, then the template horizon we have been seeking is only 10. We can copy the template of this size at every node in the lattice and thereby obtain any possible motion through workspace.

But before we can declare that we should understand a). whether the above tolerance is *good enough*, and b). whether we can indeed concatenate two sub-paths and reproduce the original path. Addressing the second question seemed a natural approach, and it is presented in Section 6.3. However, it is important to realize tha the two questions are related. The tolerance of 0% is certainly good enough, because by the argument in the beginning of this section, we can always find the two required sub-paths. However, most likely such a tolerance cannot be achieved unless we are willing to work with extremely large template radii.

Also, a non-zero tolerance would suffice and we would still be able to eliminate all perimeter paths if we were willing to *insert* special *connection* nodes by choosing convenient (x, y, θ, κ) for such nodes. They would be the ending points of the first sub-paths and the starting points of the second sub-paths. However, inserting extra nodes violates the regularity of the template and invalidates all of its useful properties.

However, it *may* still be possible to eliminate some perimeter paths under the assumption of a non-

zero tolerance and without inserting extra nodes. We may get lucky that for a perimeter path, there exist two sub-paths in the template that, when concatenated, will generate another path that has the same initial and goal poses as the original path and that will be *similar*, or not *spatially distinct* from the original path. The *Lebesgue measure* is utilized in determining the spatial distinctness [6], or in other words the measure of distinctness in the unsigned area between the two paths. However, unlike in the first two path elimination scenarios, here the existence of a pair of eliminating sub-paths is not guaranteed. Section 6.3, however shows that it is still possible for practically important templates.

Yet another approach is possible through understanding the scale dependence of the above discussion of tolerances. Consider an example where the size of the vehicle is 100cm, $\Delta = 20\text{cm}$; then 20% of that is only 4cm. Intuitively, this tolerance seems to fall well within the precision of mechanical vehicle control. Thus, if the path ideally takes the vehicle 4cm off from the node, then almost certainly the vehicle will be able to drive exactly through the node (it may end up there anyhow due to inaccuracy of control). The only exception is when the ideal path already utilizes the maximum curvature of the vehicle, in which case if we still split the original path at the node in question, then the vehicle most likely will end up slightly off from the goal node. As long as this final error is within an acceptable tolerance (or the same mapcell), this maneuver is successful. The only caution is that in concatenating infinitely many such "inaccurate" paths, the terminal error will grow without bound.

6.3 Obtaining a Minimal Set of Paths

We obtained the minimal set of spatially-distinct paths by eliminating the paths that were not distinct. We were able to obtain some remarkable results with this process. In some fairly large templates ($R_T > 10$), we were consistently able to eliminate over 98% of paths (i.e. thousands of paths). Fig. 7 shows some successful examples of such path elimination.

However, as stated above, the existence of convenient sub-paths is not guaranteed, and so it occured sometimes that paths could not be eliminated with the above approach. All of these cases seemed to indicate that the inverse-solution trajectory generator somehow lacked the capability to generate the required sub-paths. Sometimes it was related to the fact that the original path already operated at the limits of the maximum allowable value of curvature (see Fig. 8). In other instances, the reason was less clear. Later it was determined that by allowing non-zero curvature at the connection point, it was possible to achieve many more eliminations. However, there always remained at least one or two perimiter paths (out of thousands!) that still could not be eliminated. It was necessary to resort to operating with fairly high template radii in order to achieve complete elimination. Certainly, a good question to ask is whether we actually needed those several paths.

Also let us keep in mind that this work was intended to be an improvement on SRS lattice, so we continued to only consider the paths that started and ended on nodes. The paths that do not start and end on the nodes do not make much sense in this context. However, in the forward-solution approach of the next section, it is very difficult to find paths that actually end exactly on nodes.

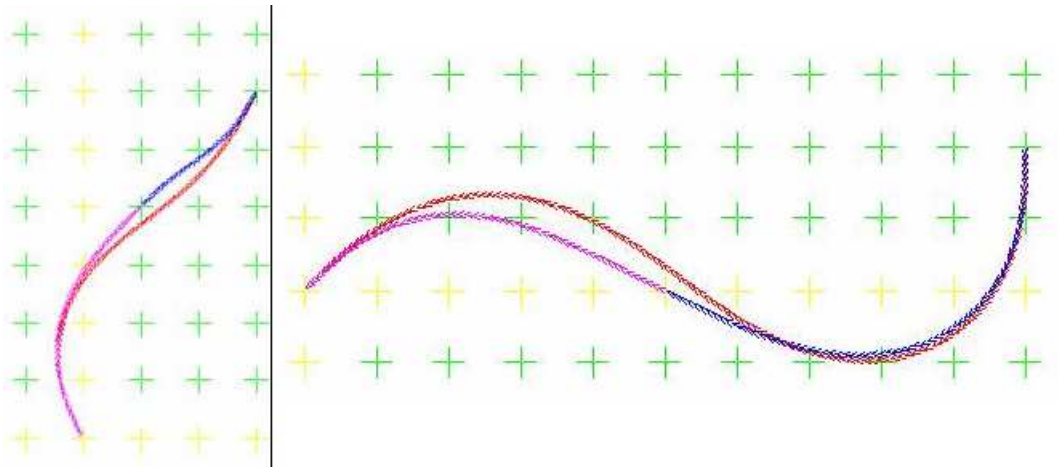

Figure 7: An example of successful path elimination. The original path is red, the two eliminating sub-paths are magenta and blue. In either case, two sub-paths *happened* to exist in the template so that a). they could be concatenated to produce a path with the same initial and final pose as the original, and b). the produced path was not spatially distinct from the original .

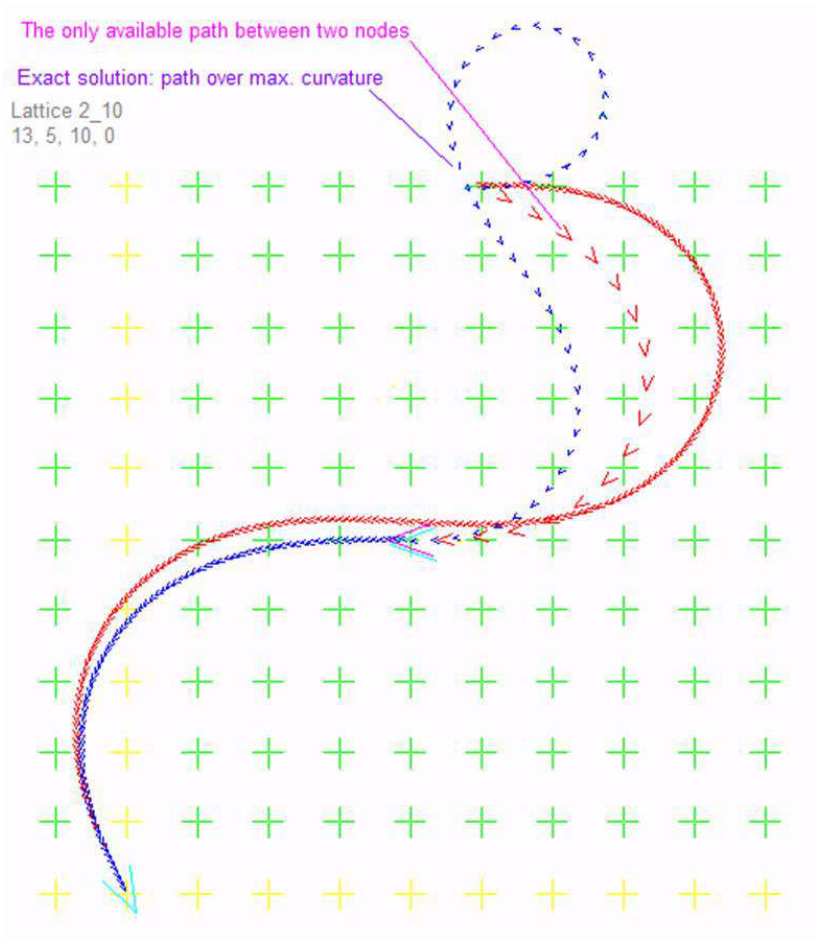

Figure 8: Elimination fails because the original path operates at near maximum curvature.

7. Forward-Solution Approach

This approach is centered around generating all possible steering functions and doing much of the above analysis in order to obtain a minimal set of spatially distinct feasible motions. By disengaging from the cubic polynomial curvature paths we hope to achieve greater generality (notably resolution-completeness). Since no constraints are implied on the steering functions we consider, outside of the inherent differential constraints of physical vehicles, we can assert that a result obtained in this fashion will necessarily be applicable to all physical vehicles. This reasoning guarantees resolution-completeness: if there exists a path that a vehicle can take, then such a path is considered by this approach (because all paths are).

As a matter of practical concern, it is very hard to enumerate all possible paths through space. By representing all feasible steering functions as branches of a tree, we were only able to enumerate branches of depth at most 9 or 10 on modern hardware (given branching factor of 5). Fig. 9 shows a projection of an example tree that has been pruned after the entire tree was created (without any pruning the tree in breadth-first manner as it is grown by eliminating redundant branches that hit the same node (obviously useless motions), we were able to get to depths of about 30 (same branching factor), as shown in Fig. 10.

One issue that comes up in generating this type of trees is the branches that manage to align themselves at a heading that is outside of any reasonable tolerances of nodes (that define "hitting" a node heading-wise). By simply keeping zero curvature, such branches can continue forever without ever hitting a node.

However, this will no longer be a problem if we require that all paths start and end at nodes, something that made a lot of sense in the inverse-solution approach. This requirement will *not* detract resolution-completeness properties of this curvature tree, because by definition of

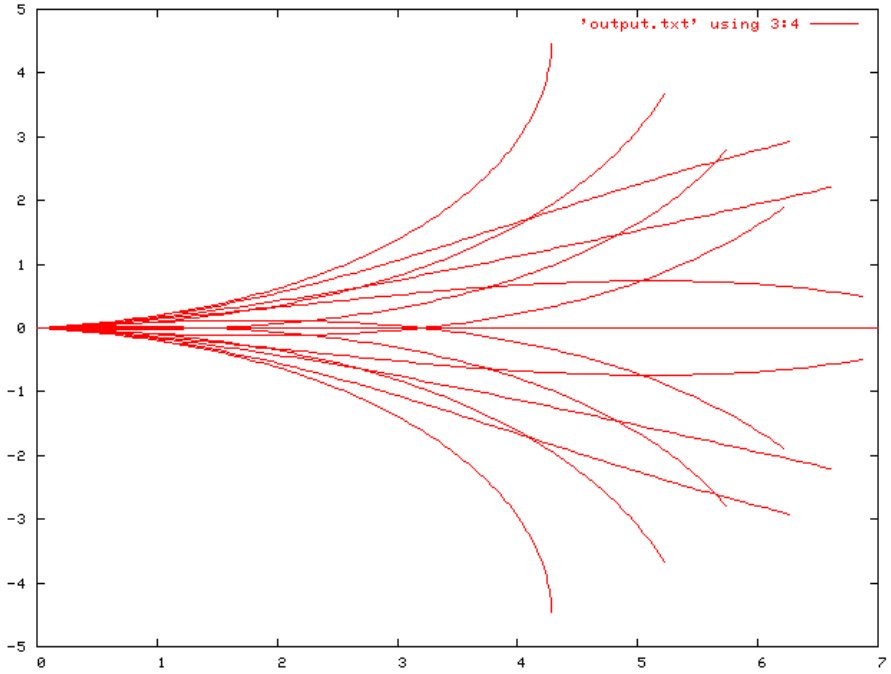

Figure 9: An workspace projection of an example curvature tree. This tree was obtained by keeping a number of most separated branches.

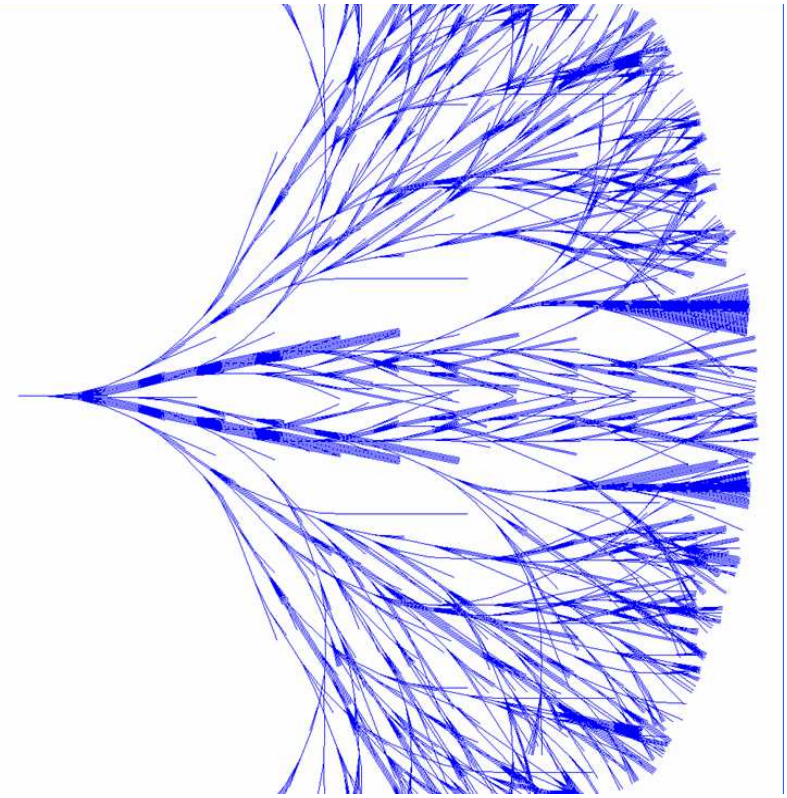

Figure 10: A projection of a curvature tree pruned during growing. This tree was pruned as it was grown by discontinuing growing branches that "hit" a node exactly and eliminating the redundant branches that hit the same node, but less precisely.

resolution-completeness the states x and $x + \varepsilon$ are considered the same for some small ε (a mapcell).

If we do make such a requirement, then, besides further decreasing the size of the tree, the main benefit of this approach over the inverse-solution one appears to lie in the ability to contain all possible paths that connect initial and goal nodes, compared to the latter that contains only unique solutions, optimal in some sense.

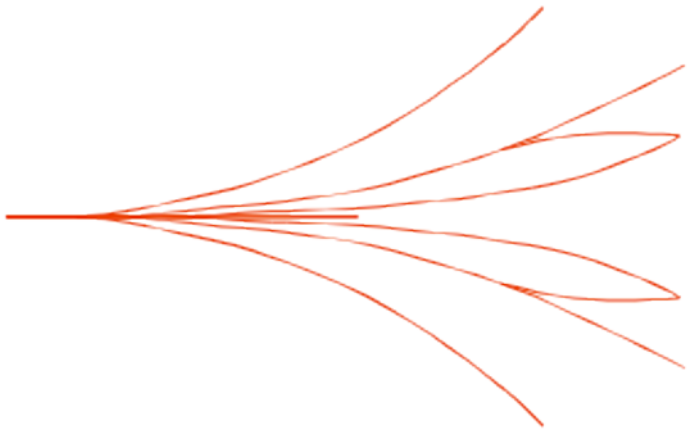

Figure 11: An example path template. This template was obtained by pruning the paths from the curvature tree.

With this in mind, the problem of proving that the inverse-solution generates paths that are very similar to what naturally emerges from the curvature tree does not appear to have much sense, since each path generated by the former corresponds to a homotopic equivalence class of paths created by the latter.

Nevertheless, it should still be possible to do a similar path elimination analysis as was done for inverse-solution paths. We accomplished this by first pruning all paths that do not terminate exactly on lattice nodes. The termination tolerance was reduced to make sure only the paths that approach a node *exactly* are preserved. This eliminated a majority of paths in Fig. 10. The paths that remained were limited to a template radius of several nodes. In this manner we obtained the template quite similar to what we got from the inverse approach (pictured in Fig. 11). This is a very important result as it shows that the forward approach produces very similar results as the inverse approach.

8. Conclusion

We have presented several important results of our study in application of polynomial curvature clothoid paths. Notably, we defined the notion of template horizon, the smallest size of a template of elementary motions that contains all “ingredient” paths to recreate all other feasible paths. The argument of existence of such a horizon is hinged on a particular choice of planning discretization (map cell size) and maximum curvature of the robot. We contend that a variety of real-word applications satisfy this condition. In addition we verified that limiting ourselves to the inverse solution paths (two-point boundary value solutions) we do not sacrifice the resolution completeness of our algorithm. This verification was accomplished by using a forward solution approach, where we enumerate all possible paths expressed as arbitrary curvature functions.

Future work mainly involves searching for a more efficient processing of clothoid paths. We would also like to efficiently re-generate the template with changing vehicle model parameters (e.g. maximum curvature). This would allow us to use this concept on a robot traveling at higher speeds: the template would adapt to the changing vehicle dynamics.

9. References

[1] P. Cheng, E. Frazzoli, S. LaValle, “Improving the performance of sampling-based planners by using a symmetry-exploiting gap reduction algorithm,” in Proc. of the IEEE/RSJ Int. Conf. on Robotics and Automation, 2004.

[2] P. Cheng, S. LaValle, “Resolution completeness for sampling-based motion planning with differential constraints,” submitted to the International Journal of Robotics Research, 2004.

[3] D. Hsu, J.-C. Latombe, R. Motwani, “Path planning in expansive configuration spaces,” in Proc. of the IEEE/RSJ Int. Conf. on Robotics and Automation, 1997.

[4] A. Kelly and B. Nagy, "Reactive nonholonomic trajectory generation via parametric optimal control," in International Journal of Robotics Research, vol. 22 (7-8), 2003.

[5] J.-C. Latombe, Robot Motion Planning, Kluwer Academic Press, Boston, 1991.

[6] S. LaValle, Planning Algorithms, Online: http://msl.cs.uiuc.edu/planning/index.html.

[7] S. Lindermann, S. LaValle, “Current issues in sampling-based motion planning,” in P. Dario, R. Chatila, ed. Proc. of the Eighth Int. Symp. on Robotics Research, Springer-Verlag, Berlin, 2004.

[8] S. Lindermann, S. LaValle, “Incremental low-discrepancy lattice methods for motion planning,” in Proc. of the IEEE/RSJ Int. Conf. on Robotics and Automation, 2003.

[9] M. Pivtoraiko and A. Kelly, “Generating motion templates for constrained motion planning in discrete configuration spaces,” submitted to the IEEE/RSJ Int. Conf. on Robotics and Automation, 2005.

[10] J.H. Reif, “Complexity of the mover’s problem and generalizations,” in Proc. of the 20th IEEE Symp. on Foundations of Computer Science (FOCS), pages 421-427, 1979.

[11] A. Scheuer, Th. Fraichard, “Planning continuous-curvature paths for car-like robots,” in Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 1304-1311, Nov. 1996.

[12] A. Scheuer, Ch. Laugier, “Planning sub-optimal and continuous-curvature paths for car-like robots,” in Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 25-31, Oct. 1998.

[13] Z. Schiller, “Dynamic motion planning of autonomous vehicles,” in IEEE Transactions on Robotics and Automation, vol. 7, no. 2, April 1991.

[14] D.H.Shin and S. Singh, “Path Generation for Robot Vehicles Using Composite Clothoid Segments,” Technical Report CMU-RI-TR-90-31, Robotics Institute, Carnegie Mellon University, December, 1990.

Index