Experimental Results for Spline Based Obstacle Avoidance of an Off-Road Ground Vehicle

John Connors and Gabriel Elkaim
Jack Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, California, 95064

BIOGRAPHY

John Connors completed B.A. degrees in Computer Science and Mathematics at the University of California at Berkeley, in 2004. He worked as a PCB engineer and consultant designing audio/video switching devices before starting as a graduate student in 2005 at the Jack Baskin School of Engineering, University of California, Santa Cruz, earning a M.S. in Computer Engineering in 2007. His research interests include electric vehicles, robotics and autonomous control. His current project focuses on outfitting an autonomous offroad ground vehicle as a reconfigurable autonomous testbed for comparative analysis of various control techniques and sensor suites.

Gabriel Elkaim received his B.S. degree in Mechanical/Aerospace Engineering from Princeton University, in 1990, and both M.S. and Ph.D. degrees from Stanford University, in Aeronautics and Astronautics, in 1995 and 2002 respectively. In 2003, he joined the faculty of the Computer Engineering department, at the Jack Baskin School of Engineering, University of California, Santa Cruz, as an Assistant Professor. His research interests include control systems, sensor fusion, GPS, system identification and autonomous vehicle systems. His research focuses on intelligent autonomous vehicles, with an emphasis on robust guidance, navigation and control strategies.

ABSTRACT

For point to point navigation, calculating suitable paths for an an autonomous ground vehicle is computationally difficult. Maneuvering an autonomous vehicle safely around obstacles is essential, and the ability to generate safe paths in a real time environment is crucial for vehicle viability. We previously presented a method for developing feasible paths through complicated environments using a baseline smooth path based on cubic splines. This method iteratively refines the path to more directly compute a feasible path and thus find an efficient, collision free path in real time through an unstructured environment. This method, when implemented in a receding horizon fashion, becomes the basis for high level control.

Previous work on spline collision free path generation is extended to include experimental validation using the Overbot, an autonomous ground vehicle developed for the original DARPA Grand Challenge in 2004. Using the previously developed software, an information grid is populated with potential obstacles. This grid becomes the basis for the pixel search along the spline path, that identifies collision points to be resolved with the presented methodology. Tests are conducted on a $600m$ offroad course using randomly generated virtual obstacles. Experimental results demonstrate good performance, with collision free paths being found, or a termination criteria reached, in under one second.

INTRODUCTION

Autonomous vehicles refer to a class of robotics in involving vehicle based systems that control their own behavior without external human input. Common in military and commercial applications, current research focuses on the use of autonomous vehicles for personal applications. In unknown or dynamic environments, trajectory generation is essential to maneuver safely from point to point. A new path planning method is implemented on one such vehicle, the Overbot, and demonstrates the ability to traverse GPS waypoints while avoiding obstacles and honoring a defined corridor constraint.

PATH PLANNING ALGORITHM

In the context of path planning, it is obvious that a path, f, needs to be continuous and, furthermore, when considering vehicles such as the Overbot, which are unable to instantaneously change heading, related to $D^1 f$, the continuity of the first derivative must also be enforced in order to guarantee a feasible path. The additional restriction that the second derivative remain continuous allows the vehicle to remain moving throughout its path. A discontinuity in $D^2 f$, which is related to steering angle, would require a vehicle like the Overbot to stop and reposition its front wheels in order to match the discontinuity. As a result, cubic splines are used to represent the trajectory, as they easily meet the required conditions. The mathematics of using cubic splines for path planning is discussed in previous work [3][4].

PATH OBJECTIVE AND COORDINATE SPACES

The Overbot was constructed to accept a sequence of GPS waypoints used to define the high level mission. The vehicle's task is to traverse through each waypoint, in order, while staying within a given radius of the nominal straight line path (Figure 1). This radius defines a corridor bounding the vehicle's possible locations.

For the detection of obstacles, the Overbot includes various imaging devices such as a color camera and laser rangefinder. As all sensors are fixed on the vehicle, obstacles are viewed relative to the vehicle position and heading, in the body reference frame. Because the vehicle is constantly in motion, the obstacles must be rotated and shifted into a constant reference frame, referred to as the Map Frame. The GPS Frame refers to the North East Down reference frame of the GPS waypoints.

INITIAL PATH SEGMENTS

The Overbot is a dynamic system with motion, control and sensor error as well as limited sensor range. It is therefore naive to pre-plan the entire vehicle path from the outset. Instead, the vehicle continually generates trajectories from the current position and orientation, using the most current obstacle map, in a receding horizon fashion. Paths are limited to consider only the next two waypoints. In this way, it is ensured that the next turn will be maneuverable and once executed, a new path will be generated to guide the vehicle into a suitable position for the following turn. For this reason, only three consecutive waypoints are used at any given time, the most recently passed, $W_1^{GPS} = (x_1^{GPS}, y_1^{GPS})$ and the following two points, W_2^{GPS} and W_3^{GPS} (Figure 1).

In order to construct a spline through these points, it must be guaranteed that $(x_1^{GPS}, x_2^{GPS}, x_3^{GPS})$ is strictly increasing [3][5]. Let W_M^{GPS} be the midpoint of the line segment between W_1^{GPS} and W_3^{GPS}. Define the Path Frame to be the GPS Frame rotated and shifted such that W_i^{GPS} maps to W_i^{Path}, W_1^{Path} is at the origin, x_3^{Path} is positive, and the line segment between W_M^{Path} and W_2^{Path} is vertical (Figure 1). Because W_M^{Path} and W_2^{Path} are vertically aligned, the waypoints are constrained not to double back, ensuring that $(x_1^{Path}, x_2^{Path}, x_3^{Path})$ is strictly increasing. A spline can now be constructed using these points. In practice, it has been found to the useful to include additional points along the straight line path. By including additional points, the initial path is kept closer to the nominal straight line path and is therefore less likely to violate the corridor constraint. The quantity and number of these points depend greatly on the relationship between the individual path lengths and the corridor width. These points are only used for the initial path estimate and are not rigid constraints in the final path.

At this point, a cubic spline, f, is computed through the points $\xi = (x_V^{Path}, x_2^{Path}, x_3^{Path})$ to represent an initial path. The spline begins at the vehicle's current position, W_V^{Path},

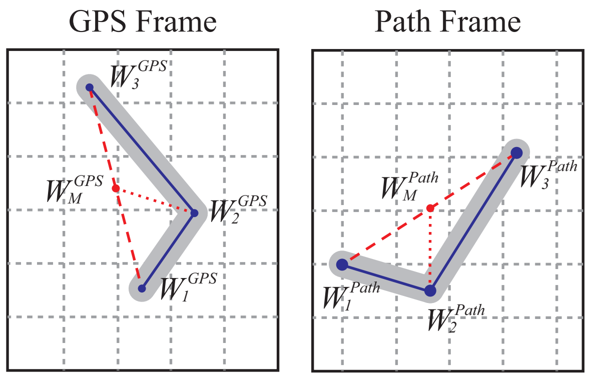

Fig. 1. The current path segments are rotated to guarantee monotonicity.

with an initial heading, C_V^{Path}, equal to the heading of the vehicle at W_V^{Path}. The final heading, C_F^{Path}, is also constrained to be tangent to the second path segment. The resulting function, f, is additionally constrained to be continuous through the second derivative, $D^2 f$.

OBSTACLE CONSTRAINTS

DETECTING COLLISIONS

On the Overbot vehicle, obstacles are mapped from various sensors into the Map Frame. The map is stored as a pixelated map, with each pixel denoting the presence or absence of an obstacle. In some applications the map may contain information about the severity of the terrain. To validate a path as obstacle free, it is evaluated within the map. A transformation $z^{Map} = H(z^{Path})$ is defined as the needed rotation, shift and scaling required to translate from the Path Frame to the Map Frame. The path, therefore, becomes $H \cdot f$ in the Map Frame, evaluated over $[H(z_V^{Path}), H(z_3^{Path})]$.

To evaluate the curve within the map, a pixelated version of the curve is generated [2]. The converted spline can be evaluated at each pixel and compared to the map to detect a collision. Since the path has no width, each obstacle must be inflated by at least half the width of the vehicle to compensate for the vehicle's size. Alternately, a corridor could be defined, having the curve of the spline but the width of the vehicle. Using this method, every pixel within the corridor would need to be evaluated instead of the single pixel wide path. The faster solution would rely on the relationship between path length and the size and density of the obstacles. In this work, obstacles are inflated when added to the obstacle map. The co-location of a path and obstacle pixel indicates a collision, each of which is recorded for later use in resolving these collisions.

PATH MANIPULATION

When the path collides with an obstacle, or violates the corridor constraint, the spline is manipulated to avoid the given obstacle. As multiple collisions may have occurred, the list of collision pixels is truncated to include only the first collision.

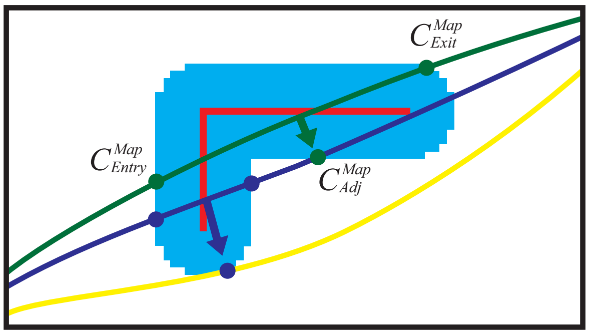

Fig. 2. The midpoint of the collision is adjusted until a clear path is found.

The first and last pixels of this collision define the entry and exit points of the collision, C_{Entry}^{Map} and C_{Exit}^{Map} (Figure 2). An additional control point is introduced into the spline to guide the path around the obstacle. The midpoint, $C_{Adj}^{Map} = (C_{Entry}^{Map} + C_{Exit}^{Map})/2$ is used as a point of manipulation. C_{Adj}^{Map} is moved perpendicular to the line $[C_{Entry}^{Map} C_{Exit}^{Map}]$ by a distance of approximately one pixel per step. The point is continually updated and checked against the map and corridor until C_{Adj}^{Map} is free of collision. Therefore, a path through C_{Adj}^{Map} will no longer collide with the obstacle at that point. C_{Adj}^{Map} is mapped into C_{Adj}^{Path} and added to the list of control points. A new spline is computed through the new set of points, retaining the desired characteristics of the previous path, but also passing through the new point, C_{Adj}^{Map}.

Each new spline is mapped back into the Map Frame and the process is repeated. If the new spline still intersects the obstacle, the collision will tend to be shorter and closer to the edge of the obstacle. Thus the algorithm will continue to displace the spline around the remaining portion of the obstacle (Figure 3). Because the points are fit with cubic functions, and x_i^{Path} is constrained to be strictly increasing, a large number of control points within close proximity to each other can cause large vertical deviations in the path. A simple adjustment to the control points is used to alleviate this issue. When adding a new point, C_{Adj}^{Path}, any other control points within a given radius of C_{Adj}^{Path} are removes. In essence, when an adjustment fails to avoid the obstacle, the new point replaces the old one. All points, except the initial position, are removable. Deletion of the endpoint is acceptable as the path will be replanned after the next way point, and the second segment of the three waypoint set will never be fully executed.

There are, however, instances when both adjacent control points may be necessary. A cyclic insertion and deletion of similar points may occur and the radius can be adjusted to allow for both points. Steps can also be taken to help avoid the situation. Though C_{Adj}^{Map} is moved in increments of approximately one pixel, a minimum radius for the first adjustment is enforced. There are cases when the spline is moved away from one section of the obstacle and onto another. The adjust found for the second collision may be to move

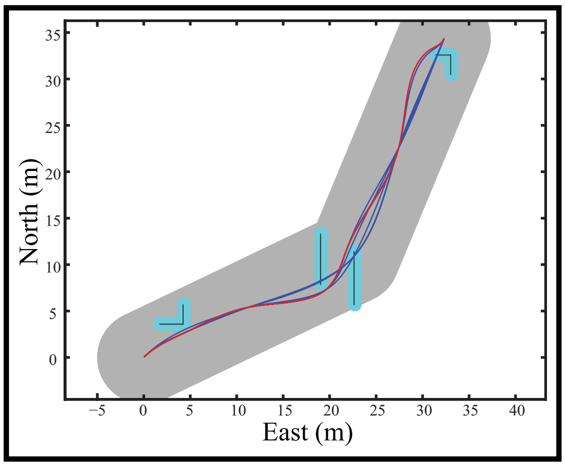

Fig. 3. Smooth curves (blue) are iteratively generated, bending around obstacles until a collision free path (red) is found.

the curve back, and a cycle will develop. By detecting these cycles, and increasing the minimum adjustment radius, larger and larger steps away from the obstacle are used, eventually breaking the cycle. In this way irregularities inherent in certain obstacles are overcome.

The final obstacle-free spline is mapped back into the GPS frame to be used by the vehicle. The path, as defined by the spline, ensures safe passage of the vehicle through its environment and contains all necessary information for vehicle control. The steering angle can be derived from the derivatives of the path. The control architecture is such that the spline path forms the feed forward term for conventional feedback used to keep the vehicle on the reference path.

ADAPTING THE ALGORITHM

A COMPILED IMPLEMENTATION

This path planning algorithm was demonstrated and analyzed previously using MATLAB [4]. However, the software and path planning run on the Overbot is compiled C code designed to run faster and more efficiently than the simulation implementation. The simulation environment used spline libraries provided by the makers of MATLAB which are not available on the Overbot platform. The calculation of the cubic splines involves solving for the coefficients of each of the polynomials involved. Because of the structure of the polynomial constraints, the coefficients are coupled and must be solved for simultaneously. Given a set of $l+1$ control points, and an initial and final heading, a system of linear equations can be established $Ax = b$, where $A \in \Re^{4\cdot l \times 4\cdot l}$ and $b \in \Re^{4\cdot l \times 1}$ contain information on the control points and derivatives and $x \in \Re^{4\cdot l \times 1}$ is a vector of the coefficients of the cubic splines. Solving these equations computationally can be accomplished more efficiently using LU decomposition than by calculating A^{-1} directly [9]. For this approach to produce a solution,

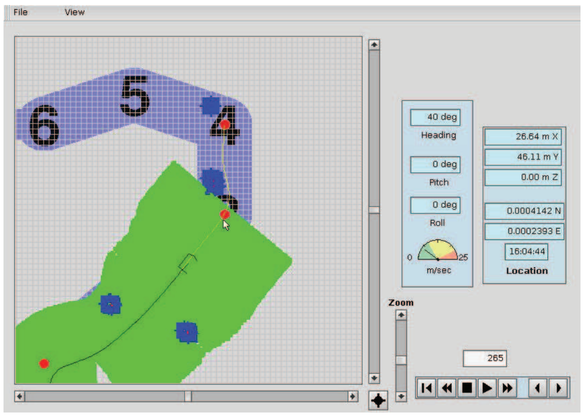

Fig. 4. The inflation of an obstacle is approximated by a square and cross. The obstacles and paths are displayed to aid development.

A is required to be full rank. This has been guaranteed by construction, however, and is discussed in previous work [3].

ELIMINATION OF THE MAP FRAME

The definition of the path planning problem discusses the transformation between the GPS and Map Frames. The Overbot's vision implementation aligns the obstacle map with a GPS North East Down (NED) coordinate system and centers both maps about the same point. The transformation from one system to the other is replaced by scalar multiplication of the map's cell resolution and reduces the computation required for the transformation, simplifying the algorithm and decreasing running time.

OBSTACLE INFLATION

The path planning techniques originally used on the Overbot evaluated an arced section about the curve for obstacles. Due to the frequency of replanning, this method requires considerably more time over other methods. To increase efficiency, an inflation radius is defined about each obstacle within the map. A circle, though ideal for inflation, requires additional comparisons to draw accurately. Instead, the circle was approximated by the use of a square and cross as demonstrated in Figure 4.

Additional optimizations were included to further reduce the time required for this procedure. As obstacles are likely to be larger than one map pixel, much of the inflation process becomes redundant. A simple check is performed on the four immediate neighbors of the obstacle cell being inflated. If a given neighbor cell contains an obstacle, then it can be assumed that both quadrants on that side of the current cell have already been inflated and can be skipped. This method only requires a few extra comparisons, but in practice can greatly reduce memory accesses to the obstacle map.

For development and debugging purposes, log files are generated by the Overbot software to track the behavior of the vehicle. These logging routines were updated to log the additional map information as well as the splines that result from the new path planning techniques. The log files can be read by a visualization program that shows, in real time, the environment as seen by the Overbot, the current path solution and tracks the vehicle's movements. This program was also modified to display the inflation cells and plot the spline paths as shown in Figure 4. This program is an extremely powerful debugging tool as it not only shows data from the vehicle, but can be used to playback log files generated from running the software on simulated environments.

ROTATION MODIFICATION

During testing of the algorithm within the architecture of the vehicle, undesired behavior was discovered. In the case of three linear, or near linear waypoints, the defined Path Frame rotation will generally rotate the segments to be nearly vertical. In these cases, the behavior of the cubic polynomials was empirically found to be irregular with vertical fluctuations resulting in the vehicle doubling back on the path. When the angle between the two path segments is greater than $\frac{3 \cdot \pi}{4}$, the rotation angle is defined to make the first path segment horizontal. Because the angle between the segments is obtuse, the monotonicity of the points is still guaranteed.

CURVATURE CALCULATIONS

The MATLAB simulations discussed previously were used only to generate paths. To be useful for autonomous vehicles, control must be implemented to follow the calculated path. The Overbot architecture uses a measure of curvature to command a desired steering angle where the curvature $c = 1/r$, the radius the resulting circle. Therefore $c = 1/r = 2\pi/2\pi \cdot r$, or more intuitively, the ratio of the change in heading to a change in arc length, $\frac{d\theta}{dL}$. Given a spline $f(x)$, the desired curvature c, therefore, is $c = \frac{d\theta}{dL} = \frac{d\theta}{ds} \cdot \frac{ds}{dx} \cdot \frac{dx}{dL}$ where $s = \frac{df}{dx}$ is the slope of the spline. The heading is a function of slope, $\theta = arctan(s)$, so $\frac{d\theta}{ds} = \frac{d}{ds} \cdot arctan(s) = \frac{1}{1+s^2}$. As $s = \frac{df}{dx}$, it follows that $\frac{ds}{dx} = \frac{d^2f}{dx^2}$. For an instantaneous slope s, $dL = \sqrt{dx^2 + (s \cdot dx)^2} = dx \cdot \sqrt{1+s^2}$, therefore $\frac{dx}{dL} = \frac{1}{\sqrt{1+s^2}}$. Hence,

$$c = \frac{d\theta}{dL} = \frac{d\theta}{ds} \cdot \frac{ds}{dx} \cdot \frac{dx}{dL} = \frac{1}{(1+s^2)^{3/2}} \cdot \frac{d^2f}{dx^2}.$$

VEHICLE PERFORMANCE

The path planning algorithm described here is implemented on the Overbot, and tested in a real world environment. For comparison, all tests are performed with the spline based approach presented here, as well as the arc based method originally implemented on the Overbot [7]. This earlier approach was found to have many weaknesses in real world tests, the results of which are available in previous work. The method relies on singular circular arcs and generally results in paths of only $10 - 20m$.

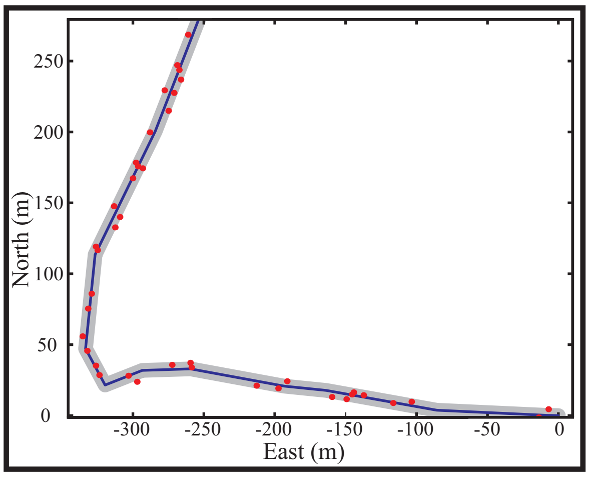

Fig. 5. The test course stretched over 600m across a hillside with randomly placed, point-like obstacles. This course contains 40 obstacles.

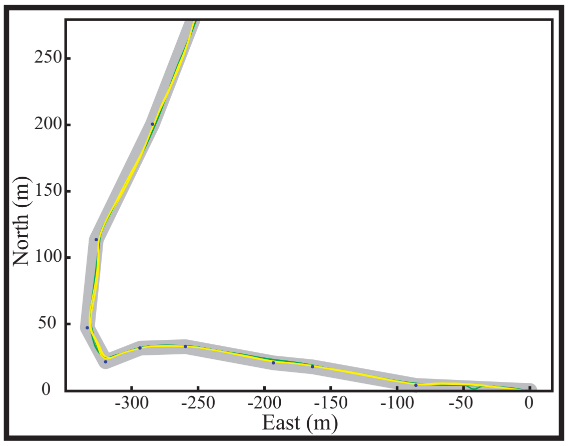

Fig. 6. Both the arc-based method (green) and the spline-based approach (yellow) were run through the course without the presence of obstacles.

TEST SETUP

The Overbot's test route consists of a dirt road in a field of tall grass and medium to large sized bushes. The course, shown in Figure 5, followed a 600m rutted dirt road with a right turn approximately halfway through. The course spans a hillside and various sections of the route are inclined, both up and down, as well as pitched side to side. The final 200m of this course represents a continual incline, climbing approximately 30m.

The vehicle's vision software was modified to load a simulated set of obstacles in order to provide controlled environments for testing. Obstacles are mapped relative to GPS coordinates and therefore provide consistent testing environments for each test run of the vehicle. Sample environments contained randomly placed, point-like obstacles of varying quantity. An example is provided in Figure 5.

For each test, the vehicle started from the same position and orientation. The entire course provided clear view of the sky for continuous GPS signal reception. The success of each test run was determined by the forward progress made and the ability to find feasible paths. The running time was recorded as the real time elapsed for the computation and verification of the path as well as the feedback terms required for control.

SUCCESSFUL PATHS

Initial tests were conducted on the previously mentioned course without the presence of obstacles. The feedback controller was refined to more accurately track the curves involved in this course. Both the original arc-based approach, and the new spline method presented here were tested. Each were able to consistently complete the defined course. This test demonstrates the functionality of the algorithm within the context of the vehicle and confirms the satisfiability of the corridor constraint.

The completion of the multiple waypoint course also indicates the successful implementation of the receding horizon control. Log files from the vehicle confirm the replanning of paths as the vehicle progressed through the course. In general, the new paths were very similar to previous routes but were able to compensate for errors in vehicle control or GPS drift. Figure 6 shows the paths actually traversed by the two planning methods, each completing the course three times. In the absence of obstacles, both algorithms performed nearly identically to each other. It should be noted that the modified controller gains greatly increased the performance of the arc-based approach. This method is now empirically more stable and more consistent than in prior experiments [7].

A virtual environment was constructed with 10 randomly placed obstacles used as a simulated environment for the Overbot. Again, both methods were run through the entire course. The spline-based algorithm was able to successfully find clear paths around the new obstacles and traverse the course. The test data, shown in Figure 7, show paths adapting to obstacles and maneuvering around them.

Because of the use of a simulated environment, all obstacles are immediately available to the vehicle. Under normal operating modes, the obstacles would not be detected until they are within range of the vision sensors. Because of this, the path planning is often considering obstacles that may not yet have been detected under normal use. As a result, the new algorithm is performing more computations than normally necessary and as a consequence, this test fails to fully demonstrate the adaptability of the algorithm to deal with new obstacles as they appear.

An additional test was run on an environment containing 20 obstacles. Due to various issues, discussed in the next section, the test was only run up to the main turn, approximately 300m. The results continue to verify the success of the algorithm on the vehicle. As shown in Figure 8, the obstacles are denser than previously tested. The vehicle is able to maneuver

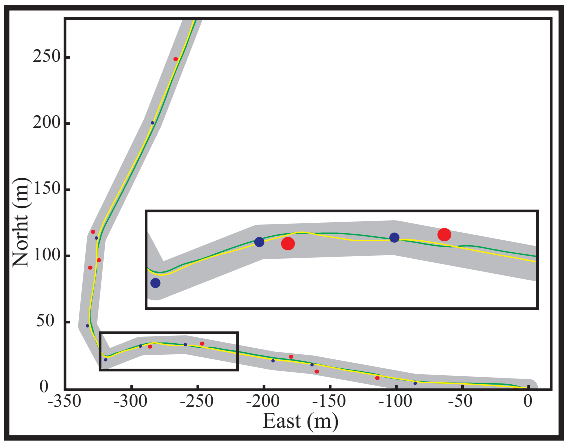

Fig. 7. The vehicle was also run through an environments with 10 randomly placed obstacles (red). The paths followed by the arc (green) and spline (yellow) methods are shown.

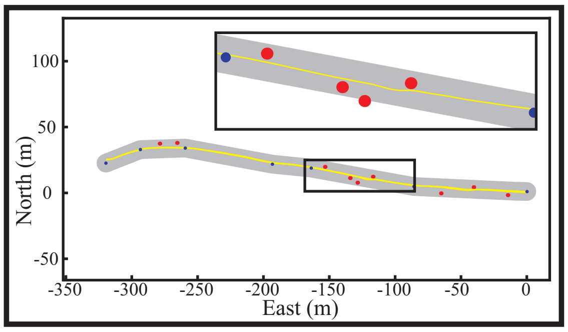

Fig. 8. The spline-based path planning presented here was tested in an environment with more densely placed obstacles.

around these obstacles, even when complex maneuvers are needed to avoid multiple obstacles. This test demonstrates the algorithm's ability to wind paths between obstacles. Again the vehicle traverses the course while satisfying the corridor constraint and the avoiding obstacles.

RUNNING TIME

The running time of the path planning algorithm was logged as the real time elapsed for the construction and verification of the path, as well as the calculation of the curvature and control terms. The new path planning method considers much greater length paths than the previous method used and generally requires more time for verification. The running time increases as the quantity and density of obstacles increases. As stated above, the testing setup made all obstacles available to the path planning algorithms even though in normal operation the limited vision range would limit the number of obstacles considered at any time. As such, it is likely that addition iterations were required, and more computation time was used during these tests than may be used normally. The arc-based approached is designed to consider only paths within the sensor range, and therefore the running time of that approach

Environment	Arc Mean	Arc Max	Spline Mean	Spline Max
No Obstacles	29.9ms	115.0ms	27.5ms	121.0ms
10 Obstacles	30.3ms	53.0ms	40.2ms	215.0ms
20 Obstacles	N/A ms	N/A ms	36.0ms	657.9ms

TABLE I
RUNNING TIMES ARE COMPUTED FOR THE ORIGINAL ARC-BASED ALGORITHM AND COMPARED TO THE NEW SPLINE-BASED METHOD FOR VARIOUS ENVIRONMENTS.

is not effected by this setup.

For the initial tests involving an obstacle free environment, the running time for both methods is comparable. Both took an average of approximately 30ms with a peak time of 120ms. For a real time vehicle, these times are within an acceptable boundary. As the environments became more complex and included the presence of obstacles, the running time of the spline-based algorithm increased. In general, the average running time remained in the $30-40$ms range, however, the maximum time taken by a single iteration increased greatly in the more complex environments. As stated above, this may be an artifact of infinite sensor range generated by the simulated obstacles. Even in the presence of complex environments, no running time exceeded 1sec. Given correct software architecture, this performance would be acceptable for autonomous vehicles. A complete set of data is available in Table I.

As testing environments became more complex, and the density of the obstacles increased, the running time of the new path planning method increased greatly. The software architecture in use on the Overbot relies on the strict timing of some events. As the running time of the path planning algorithm increased, the frequency of calls to other software decreased. The GPS software was unable to receive updates as often as the software requires and incorrectly concluded that GPS data was lost. As a result, the vehicle miscalculated its position, generated fault conditions and stopped despite the presence of feasible paths. As such, testing of more complicated environments was not possible on this vehicle. The method is still shown to be valid, but more useful within a different software structure. The use of a faster processor, the restructuring of the vehicle software, or the use of a more parallel approach to path planning would help alleviate this problem.

CONCLUSION

A new path planning algorithm is developed to run on the Overbot, an autonomous ground vehicle, within the software environment already present. Modifications are made to interface with the sensor and actuator systems present on the vehicle. A test course is set up and implementation results calculated for this algorithm on the vehicle. The results demonstrate the success of this new algorithm on the Overbot by planning paths through the defined course. The algorithm is implemented in a receding horizon fashion and uses feedback control to track the computed paths. Within the context of the Overbot, the running time of the algorithm has been shown to cause issues with other software running on the vehicle. As such, test are only conducted on simpler environments, though test data indicates the success of the algorithm in more intensive situations.

FUTURE WORK

DYNAMIC OBSTACLES

In static environments, the presented algorithm has been demonstrated to perform well for most circumstances. For many military or rescue situations, it may be acceptable to assume a fixed environment. In the case of urban applications, however, the vehicle must interact with other vehicles and pedestrians. Without universal algorithms or vehicle communication, the behavior of other vehicles may be unpredictable. Furthermore, without knowledge of other vehicles, it may be difficult for vision sensors to differentiate between static and dynamic obstacles. For these reasons, less confidence can be placed on the accuracy of the obstacle map.

As this algorithm is continually recalculated using up to date information, it is theoretically possible to avoid dynamic obstacles by increasing the rate at which a path is computed. This refresh rate would depend on the speed of the vehicle as well as the speed of the obstacles. In practice, it would likely be too fast to compute in real time. Furthermore, in order to be useful, the sensors would need to reevaluate the entire environment between each iteration of the algorithm. The slow speed of the vision techniques and the high frequency of the refresh rate would ultimately make this approach infeasible.

A more practical approach may be to estimate the movement of each obstacle. By computing a trajectory for the various dynamic obstacles, estimates of future position could be used when detecting potential vehicle collisions. The path would be evaluated within estimated, timed versions of the obstacle map. The approach would require time or speed information to be encoded into the path planning algorithm. The implementation presented in this paper plans only the vehicle position and orientation. Some evaluation may be needed to determine the degree of accuracy needed to accurately track obstacles. Obstacles that are farther away from the vehicle may have a large change in position as well as be more likely to stray from a straight line course. However, this technique need only overcome the short falls of the slower refresh rate. To that end, it may be likely that assuming straight line courses for those short periods of time will yield sufficient results. On the Overbot, the CCD camera could be used to track the motion of an obstacle, while the Lidar accurately maps the given obstacle's location.

BOUNDED DERIVATIVES

By construction, the splines used in this algorithm are able to enforce continuity on the second derivative, but unable to bound their value as is possible in some other work [6]. Most ground based vehicles have physical limits on the steering systems that limit the turning rate of the vehicle. Because the algorithm does not enforce bounds on the second derivative, it is possible to construct paths that are not traversable. This issue is further complicated by the fact that the second derivative represents the rate of change of the spline's slope, not heading. Therefore, the needed bound is a function of the first derivative at the given point.

ACKNOWLEDGEMENTS

This work was supported by NSF grant 0521675. We would like to thank Team Overbot for all the time and energy they put into the construction of the Overbot, and John Nagle for his generosity and continuing support of the vehicle and project. Special thanks go to Scott Triglia from Harvey Mudd College for his help in testing and debugging.

REFERENCES

[1] T. Berglund, H. Jonsson, and I. Sderkvist, "An obstacle-avoiding minimum variation b-spline problem," *Proceedings of the 2003 International Conference on Geometric Modeling and Graphics*, 2003.

[2] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, p. 25, 1965.

[3] J. Connors and G. H. Elkaim, "Manipulaiting b-spline based paths for obstacle avoidance in autonomous ground vehicles," *Proceedings of the ION National Technical Meeting*, 2007.

[4] J. Connors and G. H. Elkaim, "Analysis of a spline based, obstacle avoiding path planning algorithm," *Proceedings of the Vehicular Technology Conference*, 2007.

[5] C. de Boor, *A Practical Guide to Splines*, revised ed. Springer, 1978.

[6] H. Delingette, M. Hebert, and K. Ikeuchi, "Trajectory generation with curvature constraint based on energy minimization," *International Workshop on Intelligent Robots and Systems*, p. 206, 1991.

[7] G. Elkaim, J. Connors, and J. Nagle, "The overbot: An off-road autonomous ground vehicle testbed," *Proceedings of the ION Global Navigation Satellite Systems Conference*, 2006.

[8] Y. Kanayama and B. I. Hartman, "Smooth local path planning for autonomous vehicles," *International Journal of robotics Research*, vol. 16, no. 3, p. 263, 1997.

[9] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge University Press, October 1992.

[10] Z. shiller and Y.-R. Gwo, "Dynamic motion planning of autonomous vehicles," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 2, p. 241, 1991.

[11] S. Thompson and S. Kagami, "Continous curvature trajectory generation with obstacle avoidance for car-like robots," *Proceedings of the 2005 International Conference on Computaional Intelligence for Modelling, Control and Automation and International Intelligent Agents, Web Technologies and Internet Commerce*, 2005.

[12] C.-H. Wang and J.-G. Horng, "Constrained minimum-time path planning for robot manipulators via virtual knots of the cubic b-spline functions," *IEEE Transactions on Automatic Control*, vol. 35, no. 5, p. 573, 1990.