Analysis of a Spline Based, Obstacle Avoiding Path Planning Algorithm

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

***Abstract*—The Overbot is one of the original DARPA Grand Challenge vehicles now being used as a platform for autonomous vehicle research. The vehicle, equipped with a complete actuator and sensor suite, provides for an extremely capable robotic platform with computing infrastructure and software framework already in place to create a reconfigurable testbed.**

For point to point navigation, calculating suitable paths 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 is able to iteratively refine 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. In this work we perform Monte Carlo simulations to validate algorithm performance. The algorithm demonstrates a high success rate for all but the toughest of environments.

***Index Terms*—Mobile robot motion-planning, Motion-planning, Mobile robots, Land vehicles, Road vehicle control, Road transportation, Spline functions**

I. INTRODUCTION

The Overbot, originally designed to run the DARPA Grand Challenge, has been retasked as an off-road autonomous vehicle testbed. The vehicle is presently being used to further research in autonomous vehicles at the University of California, Santa Cruz. Various elements are being improved and upgraded including substantial efforts to improve path planning algorithms. In addition to obvious military applications, autonomous vehicles have generated significant interest for commercial applications. The removal of human interaction from everyday driving could lead to safer roads, fewer accidents and increased traffic bandwidth. Even basic vehicle operation requires significant path planning to provide an accurate, feasible and smooth path from point to point through a potentially complex environment. The introduction of obstacles, such as buildings, pedestrians and other vehicles increase the complexity and limit traversable paths.

This paper focuses on analyzing the performance of the path planning algorithm detailed in previous work [2]. The algorithm employs cubic splines to define the vehicle path and can therefore guarantee the needed conditions for a ground vehicle, namely a continuous heading and steering angle. Through the introduction of additional control points, the shape of the spline is manipulated to fall within a given corridor constraint and bend around obstacles. We have demonstrated successful path generation through a variety of environments while avoiding obstacles and the ability to compute paths in near-real time which makes this method practical for autonomous vehicle control.

We will define the path planning problem and provide a brief description of the algorithm. A more complete definition and a review of splines can be found in our previous paper [2]. We present the application of Monte Carlo simulations to our algorithm and derive algorithm performance analysis from the results.

II. 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, $D^1 f$, we must also enforce the continuity of the first derivative 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 of $D^2 f$ would require a vehicle like the Overbot to stop and reposition its front wheels in order to alter its steering angle, which is proportional to $D^2 f$.

A. PATH OBJECTIVE AND COORDINATE SPACES

The Overbot was constructed to function according to the guidelines of the DARPA sponsored Grand Challenge. The vehicle, therefore, accepts 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. Because the vehicle is constantly in motion, the obstacles must be rotated and shifted into a constant reference frame which we refer to as the Map Frame.

B. INITIAL PATH SEGMENTS

The Overbot is a dynamic system with motion, control and sensor error as well as limited sensor range. It is therefore

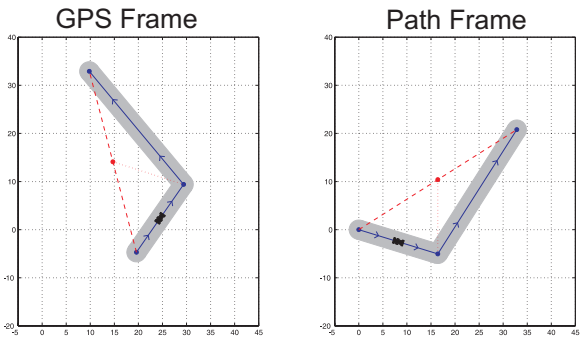

Fig. 1. The current waypoints (blue) and corridor (grey) must be rotated to guarantee that the points are monotonic.

naive to pre-plan the entire vehicle path from the outset. Instead, we replan the path from the the current position and heading, and using the most current obstacle map and do so in a receding horizon fashion. We limit our path look ahead to the next two waypoints only. In this way, we help ensure 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, we only work with three consecutive waypoints at a 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, we must guarantee that $(x_1^{GPS}, x_2^{GPS}, x_3^{GPS})$ is strictly increasing. We let W_M^{GPS} be the midpoint of the line segment between W_1^{GPS} and W_3^{GPS}. We now 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). We constrain the waypoints to not double back, therefore $(x_1^{Path}, x_2^{Path}, x_3^{Path})$ is strictly increasing. Though we can now construct a spline using these points, we have found it useful to include additional points along the straight line path. The quantity and number of these points depend greatly on the relationship between the individual path lengths and the corridor width. 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. These points are only used for the initial path estimate and will not be rigid constraints in the final path.

At this point we can apply our spline function in order to generate an initial path. We construct a path, f, of degree 3 through the points $\xi = (x_V^{Path}, x_2^{Path}, x_3^{Path})$. The spline started at the vehicle's current position, W_V^{Path} and having an initial heading C_V^{Path}, equal to the heading of the vehicle at W_V^{Path}. We can also constrain the final heading C_F^{Path}, to be tangent to the second path segment.

III. OBSTACLE CONSTRAINTS

A. DETECTING COLLISIONS

As described above, obstacles are mapped from various sensors into the Map Frame, which maintains a constant rotation and shift relative to the GPS Frame. The map is stored as a pixelated map, each pixel denoting the presence or absence of an obstacle (binary), or the severity of the terrain as required by the mission. To check our path against the map, we define a transformation $z^{Map} = H(z^{Path})$ to be the needed rotation, shift and scaling required to translate from the Path Frame to the Map Frame. Therefore our path in the Map Frame become $H \cdot f$ evaluated over $[z_V^{Path}, z_3^{Path}]$.

To evaluate our curve against the map, we create a pixelation of the curve. The converted spline can be evaluated at each pixel and compared to the map to detect a collision. It should be noted that each obstacle must be inflated by at least half the width of the vehicle to guarantee that a collision does not occur. One could also define a corridor having the curve of the spline but the width of the vehicle. Using this method, one would have to then evaluate every pixel within the corridor instead of the single pixel path. The faster solution would rely on the relationship between path length and the size and density of the obstacles. In this work, we inflate our obstacles within the map. The co-location of a path and obstacle pixel indicates a collision, which is stored for later use.

B. PATH MANIPULATION

Once it has been determined that the path collides with an obstacle, or violates the corridor constraint, we manipulate the spline to avoid that obstacle. As multiple collision may have occurred, we first edit the list of collision pixels to include only the first collision. The first and last pixels of this collision define the entry and exit points of the collision, C_{Entry}^{Map} and C_{Exit}^{Map}. We add an additional control point to our spline to guide the path around the obstacle. We use the midpoint, $C_{Adj}^{Map} = (C_{Entry}^{Map} + C_{Exit}^{Map})/2$ as our point of manipulation. We move C_{Adj}^{Map} 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. This means that a path through C_{Adj}^{Map} will no longer collide with our 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 these points still having our desired characteristics but also passing through the new point (Figure 2).

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 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, along with the constraint that x_i^{Path} be strictly increasing, a large number of control points within close proximity to each other can cause large deviations in the path. To avoid this, we include a simple adjustment to the control points; when adding a new point, C_{Adj}^{Path}, we remove any other control points within a given radius of C_{Adj}^{Path}. 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

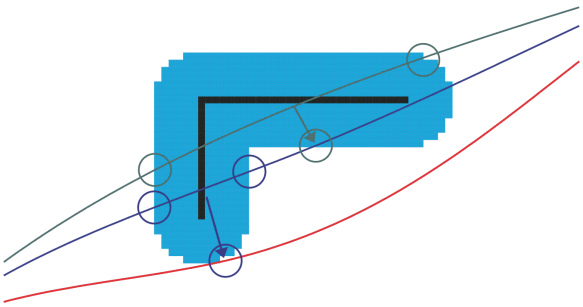

Fig. 2. The midpoint of the collision is moved out until a clear path is found.

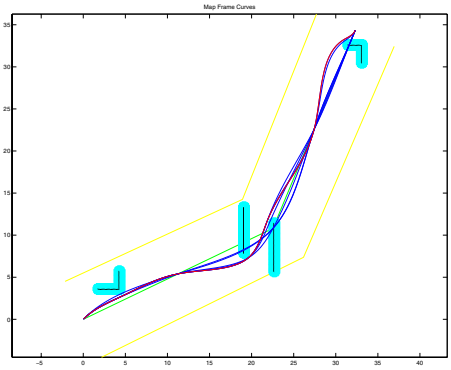

Fig. 3. We iteratively generate smooth curves(blue) bending them around obstacles until a collision free path(red) is found.

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. It is easy in these cases to detect the cyclic insertion and deletion of points and the radius can be adjusted to allow for both points. Another method, which we use, to determine when to keep the adjacent control points is as follows. Though C_{Adj}^{Map} is moved in increments of approximately one pixel, we also enforce a minimum radius for the first adjustment. This addresses situation when the spline is moved away from one section of obstacle and onto another. If the adjustment for the second segment is to move the curve back, a cycle will develop. By detecting these cycles, and increasing the minimum adjustment radius, we take larger and larger steps away from the obstacle, eventually breaking the cycle. In this way we can overcome the irregularities inherent in certain obstacles.

Once the final obstacle-free spline is found, it can be 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. The steering angle can be derived from the second derivative of the path. The control architecture is such that the spline path forms the feed forward part of the conventional feedback used to keep the vehicle on the reference path.

IV. PERFORMANCE ANALYSIS

To analyze the performance of this algorithm, we break it into the following steps: Calculation of the initial spline, evaluating a given spline, and generating new control points. The initial baseline spline is calculated only once and always through a fixed number of points and therefore requires a constant amount of time. The evaluation of a spline for obstacle collisions and corridor violations is based on the length of the spline, which is roughly proportional to the the length, L, of the two segments. Likewise, as paths tend to run down the length of the corridor, the amount of work required to adjust a control point is roughly proportional to the width, W, of the corridor. Given a limit, I, on the overall number of iterations the algorithm performs, this method is order $\bigodot(I(L+W))$.

A. MONTE CARLO SIMULATION SETUP

We perform a series of Monte Carlo simulations over random environments to determine average running times and success rates of this algorithm. For each iteration of the simulation, we generate a random path consisting of two segments, 55m and 50m long respectively, and having a random angle between them measuring in the range $(-150°, 150°)$. Longer paths would fall outside the sensor range of most vehicles and would not contribute to the difficulty of the problem. The vehicle is placed at the beginning of the first segment with a random heading in the range $(-30°, 30°)$ relative to the first segment. Obstacles are chosen from a library of configurations and randomly placed within the 5 meter corridor around the path. The first 5 meters of the corridor are left vacant to allow the vehicle to maneuver before entering the obstacle field. This simulates the receding horizon implementation of the algorithm that would have already allowed the vehicle to approach this segment from a favorable direction. Example environments are demonstrated in Figure 5.

Each iteration is performed on a new path and random placement of obstacles. An iteration of the algorithm is considered successful if a feasible path is found from the vehicle's current position to the end of the second corridor while satisfying the corridor constraint over the entire path and not colliding with any obstacles. We also measure the real time performance of each iteration as the time elapsed from the creation of the initial spline to the verification of the final path. We do not include the time needed to inflate the map as it is assumed that all obstacles can be inflated at the time of their insertion in to the map. All tests were performed on a Pentium Mobile class processor clocked at 2 GHz, using Matlab.

B. MONTE CARLO ANALYSIS

We performed a series of 5,000 iterations of the algorithm varying the number of obstacles from 5 to 50, in increments of 5, as well as varying the size of the obstacles up to several

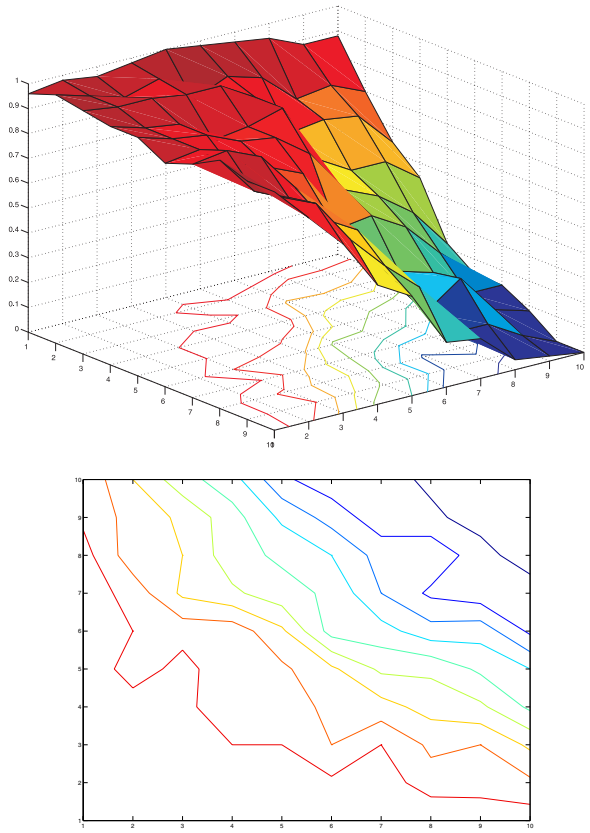

Fig. 4. Monte Carlo simulations were run over environments having between 5 and 50 obstacles of size up to several square meters to determine the success rate of the algorithm.

square meters. The algorithm was set to generate up to 50 different paths for each scenario before failing. The results of this simulation are presented in Figure 4.

The algorithm was found to be over 70% successful for most environments up to approximately 30 obstacles measuring 6 square meters, including environments that may not have been feasible at all. For more difficult environments, performance diminished quickly and the algorithm was unable to find feasible paths beyond this level of difficulty. A random sampling of these environments demonstrates their complexity (Figure [5), and in many cases confirms that no feasible path in fact exists. This method is well suited for most environments that are encountered in autonomous vehicle applications. In applications that regularly traverse difficult environments, other methods can be used that guarantee feasible paths if in fact one exists. For all other applications, this method has been shown to be very successful, and when a feasible path is not found, additional methods such as the A^* algorithm can be implemented to overcome the local difficulties.

C. RUNNING TIME

In addition to the success of each iteration, the running times were averaged for each combination of obstacle size and density. The measurements indicate the real time that elapsed

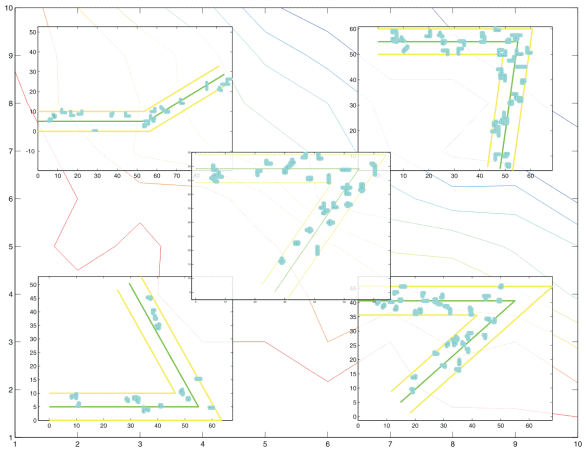

Fig. 5. Environments with increasingly large and more densely packed obstacles become difficult to traverse and some have no feasible paths.

from the the creation of the initial spline to the verification of the final path. The results follow a trend similar to that of the success rate data. The running time of the algorithm increases with the complexity of the environment until the feasibility boundary, at which time the running time reaches the $\bigodot(I(L+W))$ estimate (Figure 6). The data shows that even for complicated situations, the algorithm is able to find a feasible path, valid for the next 100m, within approximately 7 seconds. Furthermore, for the majority of applications and scenarios, solutions are available in approximately 3 or 4 seconds. In the situations in which the algorithm fails in the presence of a feasible path, more complicated methods can be implemented and the lost time is bound by $\bigodot(I(L+W))$, in this case approximately 10 seconds.

V. CONCLUSIONS

We have presented a method for solving path planning problems by bending smooth cubic splines to avoid obstacle collisions. We have developed a simple method for generating maneuverable paths through potentially complex environments (Figure 3). In the context of real time systems, this approach is designed to run efficiently by converging toward feasible solutions instead of investigating the entire path space. The use of cubic splines guarantees continuous vehicle heading and turning angle which is ideal for many ground based vehicles.

We have analyzed the performance of this algorithm through a number of randomly generated environments. The Monte Carlo simulations indicate a high success rate for the most common scenarios through very complex environments. Further analysis demonstrates running times sufficient for real time environments and suffering no great penalties in the harshest situation where other methods may need to be employed.

VI. FUTURE WORK

It has not been proven that the algorithm can guarantee a feasible path if one exists. We plan further work to investigate this topic and make the method more robust in difficult

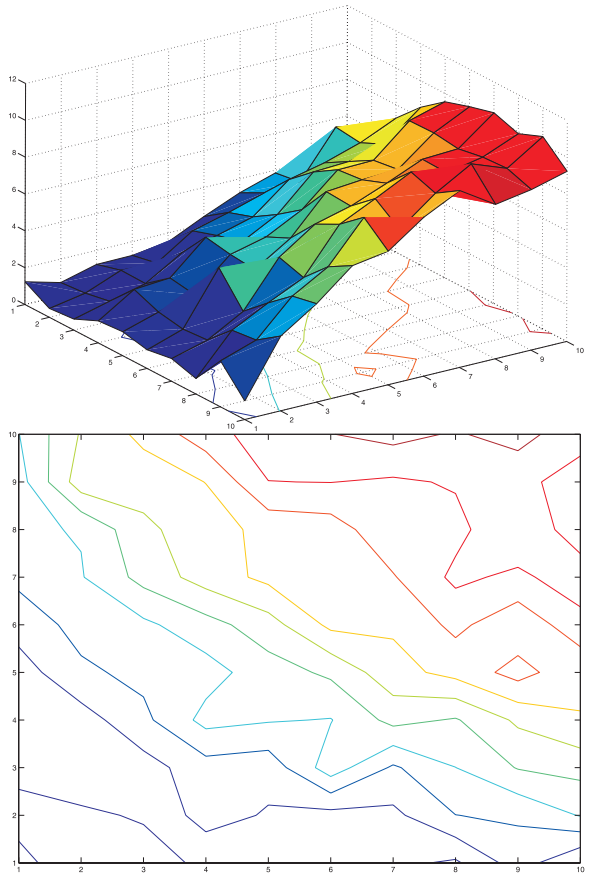

Fig. 6. The running time of the algorithm increased with the number and density of obstacles.

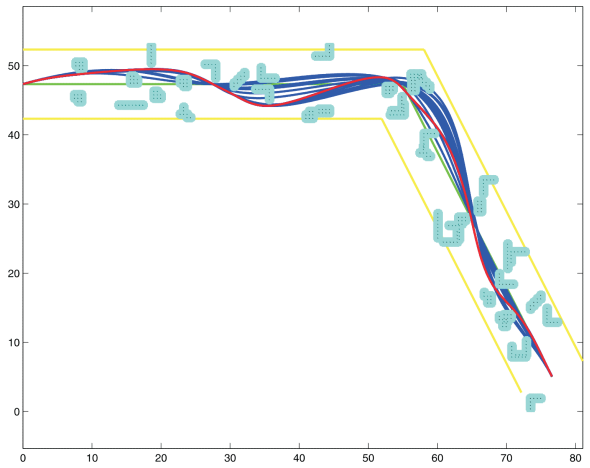

Fig. 7. The algorithm is able to find feasible paths through very difficult environments.

environments. It can be noted that a heading can also be constrained at any point on the curve by simply breaking the curve into two sections, thereby allowing the new endpoint to have first derivative constraints. This feature makes it possible then to not only force the vehicle through a narrow opening, but control its direction through that opening. Doing so, however, loses the continuity of the second derivative at that point, and thus requires a vehicle stop to readjust its steering angle. Note that in practice, the control system feedback would simply treat the discontinuity as a plant disturbance, and regulate back to the path as required. We also plan to explore the effects of moving obstacles using this method; there are interesting tradeoffs that develop due to the relationships of vehicle and the dynamic obstacle speed (note that refresh rates of both the obstacle map and the path itself become relevant with these tradeoffs).

Finally, experimental validation of this algorithm will be implemented on the real time OS of the Overbot to perform its path planning. As stated previously, planning past the next two waypoints is wasteful, unless the waypoints are very closely spaced. As the vehicle passes the first of these two waypoints, a new path is generated from its current position and proceeding two more waypoints ahead. The portion between the first and second waypoints may or may not remain the same, but will now compensate for any changes that need to be made to navigate around the second waypoint as opposed to arriving there. Furthermore, continually replanning along the segments can help compensate for positioning errors and allow for the discovery of new obstacle information.

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. 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.

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

[4] 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.

[5] 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.

[6] P. M. Kinney, M. Dooner, J. Nagel, and P. G. Trepagnier, "Kat-5: Systems based on a successful paradigm for the development of autonomous ground vehicles," *Position, Location, and Navigation Symposium*, p. 378, 2006.

[7] K. H. Lim, "Position estimation for team overbot," *Stanford University*, 2003.

[8] 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.

[9] 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.

[10] 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.

[11] A. R. Willms and S. X. Yang, "An efficient dynamic system for real-time robot-path planning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 36, no. 4, p. 755, 2006.