The **Culebra** algorithm for path planning and obstacle avoidance in Kat-5

Jorge Nagel, Paul G. Trepagnier, Cris Koutsougeras*, Powell M. Kinney, Matthew Dooner

GrayMatter, Inc.
2612 Severn Ave.
Metairie, Louisiana 70002

*Dept. of Computer Science & Industrial Technology
Southeastern LA University 10847
Hammond, LA 70402

**Abstract**

Kat-5 was the fourth vehicle to successfully finish the DARPA 2005 Grand Challenge; the first time ever that autonomous vehicles were able to successfully complete such a task. In this paper, we describe the methods that were used to develop the vehicle’s path planning and obstacle avoidance algorithms, which allowed Kat-5 to successfully navigate completely autonomously the 132 miles course over rough and previously unrehearsed and unknown terrain at relatively high speeds.

1. **Introduction**

The DARPA Grand Challenge was a competition aiming to foster the development of autonomous vehicles, which could traverse challenging and unknown terrain at relatively high speeds. Out of the original 195 vehicles entered, Team Gray’s Kat-5 was one of five vehicles to complete the course and one of only four to accomplish this feat within the 10 hour time limit. In the following text, we briefly describe the apparatus of the vehicle but the primary focus is on the algorithms for path planning and obstacle avoidance that allowed the vehicle to successfully complete the 132-mile desert course.

To set the stage, let us summarize here the rules of the race, which consequently defined the constraints of the problem. The route was initially unknown; not just unrehearsed, but indeed unknown. At the starting line, DARPA released the route to be followed 2 hours before the start of the race. They provided the route as a sequence of GPS waypoints and for each waypoint there was a tolerance associated with it, usually about 10ft. Each vehicle had to hit the waypoints within the respective tolerances or it could be disqualified. The route of 132 miles had to be completed in less than 10 hours. The route contained parts of very rough and rocky terrain, a dry lakebed, and even tunnels in which GPS reception was impaired. All motion planning and control had to be decided and executed based on real time sensory information by the car’s onboard computer without any other intervention. DARPA had guaranteed that there would exist a passable path at all times although there were obstacles to be detected and avoided. The car’s own computer had to figure out the proper maneuvering in order to proceed.

The car was a Ford Escape hybrid, which was outfitted with a commercial drive-by-wire system designed by Electronic Mobility Controls for physically impaired drivers. The hybrid’s electrical system was used to obtain power for the onboard equipment. A network of two marine computers (regular PCs but running on 12 volts) and two Mac Mini’s were used for sensor data processing and AI. The drive-by-wire system was connected to one of the PC’s via an I/O board so that the PC provided the signals to

![Figure 1: Kat-5 as it was entered in the Grand Challenge 2005 race. The two LIDAR units and the two antennas of the GPS/INS are visible on top of the solar panels.](image-url)
the drive-by-wire system. This is how the car was made drivable by our software. There was also an integrated GPS/INS unit made by Oxford Technical Solutions which provided awareness of location and motion details at all times. To sense the environment, we used two LIDAR units, which were mounted on the top of the vehicle and were actively scanning the terrain ahead of the vehicle. The readings of the LIDAR scanners were combined with those of the GPS/INS in order to create a geospatial representation of the environment relative to the vehicle. Thus, all sensor measurements were translated to a coordinate system centered on the vehicle. In this way, the onboard computers had a representation of the environmental state and they could control the actuators that drove the car. What we will analyze in the following is how these computers determined the path to follow that satisfied the constraints and avoided obstacles.

2. The Methods

To create a path planning and obstacle avoidance system for the DARPA Grand Challenge, we considered several different candidate methods. Our first implementation was a purely reactive approach. It consisted of several different behaviors. The primary behavior followed the route. It did this by “seeking” the next waypoint on the path. When the vehicle approached within a certain radius of the target waypoint, that waypoint was considered attained, and the vehicle switched to the next waypoint. The other main behavior was the obstacle avoidance behavior. This behavior avoided any obstacles directly in the vehicle’s path by steering either to the left or to the right of the given obstacle, and then gradually drifting back to the proper path. This approach was successfully used up until the DARPA site visit, where Kat-5 successfully avoided all obstacles on a 200-meter route while being evaluated by two members of DARPA. After the site visit, we decided that the algorithm was not robust enough for our needs, and the purely reactive paradigm was discarded.

At this point, we decided to use a hybrid deliberative/reactive approach. There were several choices to consider for the deliberative portion of the path planning and obstacle avoidance system. The methods that were considered for the deliberative path planning portion of the hybrid paradigm were the A* algorithm and its variants. A* is a popular algorithm in the AI field, and therefore has many different variants that were created to solve different problems. We considered the basic A* algorithm along with the Dynamic A* (D*) algorithm. To build the graph required by A*, we represented the environment as a grid of 20 cm by 20 cm cells containing binary values. Each grid cell was marked as either valid or invalid, and each grid cell contained edge transitions to each of its eight neighbors. Cells completely outside of the corridor were marked invalid, and so were any cells that contained obstacles.

After many field tests of the A* family of algorithms, we decided that they were not the best algorithms for our needs. Therefore, rather than using either A* or D* for the deliberative portion of the path planning system, instead we created our own deliberative system named “Culebra”. The inner workings of Culebra are discussed in the following section.

3. The Culebra Path Planning Algorithm

As previously mentioned, the course was defined using an RDDF containing the waypoints of the race, as well as the lateral boundary offset for any given segment. The waypoints and their corresponding lateral boundaries constrain the areas through which the vehicle can travel. Therefore, it is up to the vehicle to determine a desired path by interpreting the data coming from the onboard sensors.

3.1 Path Planning

The obvious first choice for generating a path is to try driving down the centerline of the RDDF waypoints. This centerline is generated by first creating a dense path, which is based on the RDDF with additional equidistant control points inserted between the original waypoints. A cubic B-spline [3] is then drawn using the dense path as the control points. A B-spline is a parametric curve composed of basis B-splines of degree n defined by (1).

$$S(t) = \sum_{i=0}^{m+1} P_i b_i(t)$$

with $b_i(t)$ defined as:

$$b_{i,0}(t) = \begin{cases} 1 & \text{if } t_j \leq t < t_{j+1} \\ 0 & \text{otherwise} \end{cases}$$

$$b_{i,n}(t) = \frac{t - t_j}{t_{j+n} - t_j} b_{i,n-1}(t) + \frac{t_{j+n} - t}{t_{j+n+1} - t_{j+1}} b_{i+1,n-1}(t)$$

Thus, all sensor measurements were translated to a coordinate system centered on the vehicle.
and where $P_i$ are called control points or de Boor points. The $m$-$n$ basis B-splines of degree $n$ can be defined using the Cox-deBoor recursion formula (2). For a cubic B-spline, the form for a single segment can be written as in (3):

$$S_i(t) = \sum_{k=0}^{3} P_{i-3+k} b_{i-3+k, 3}(t)$$  \hspace{1cm} (3)

where $S_i$ is the $i$-th B-spline segment and $P$ is the set of control points. Figure 2 shows a centerline path that has been generated using the RDDF of an oval track. The original RDDF waypoints are shown in yellow. The blue points represent the dense path that has been generated by creating cubic B-splines through the control points. Notice that the spline does not necessarily pass directly through each control point.

Figure 2: Centerline path generated from RDDF.

It can be seen that part of the RDDF waypoints do not have the dense path running through them. This is because only a segment of the entire RDDF is loaded at any given time. The Grand Challenge contained a massive RDDF file, too large to load into memory all at once, therefore only a section of the course was analyzed at any given moment. As the vehicle progressed through the race, more waypoints were loaded in the queue while past waypoints that were no longer relevant were dropped. After properly defining the master spline (a spline following the center of the RDDF), an algorithm was used to check for any obstacles that might prevent the vehicle from traveling through that path. The algorithm, nicknamed *Culebra* (Spanish for “snake”), aimed at shifting the control points defining the spline until no obstacles were found along the resulting path. As explained previously, two LIDAR units detect any obstacles ahead of the vehicle and enter them in a database. This database is used to check the segments between the control points for obstacles that may lie within the same geospatial region. Each check is made by generating a rectangular area, or *region of interest* (ROI), consisting of the current segment length and width as much as the width of the car plus a safety factor as shown in Figure 3.

![Figure 3: Path segments are checked for obstacles using rectangles.](image)

The ROI is then checked for obstacles found in the database. If an obstacle is found within the ROI, the segment is marked as invalid, and its control points are moved a distance $\Delta x$ in the direction normal to the path. The control points found in the vicinity of the invalid segment are also shifted in order to maintain path continuity. A normal distribution, centered at the invalid ROI, is used to determine the shift of adjacent control points.

In the first iteration, the control points are shifted $\Delta x$ to the right of the path and the obstacle check is performed at this new location. If the obstacle still has not been cleared, the following iteration shifts the control points $\Delta x$ to the left of the path, and the obstacle check is made once again. If neither the right nor left shifts yield a clear path, the shift distance is increased to $2\Delta x$ and the algorithm is run again with the new shift distance. The process is repeated while increasing the shift distance to $(n + 1)\Delta x$ with each pass. Figure 4 shows how the control points would shift in order to avoid an obstacle. The step size, $\Delta x$, was estimated using a simulator. A distance of 0.2m was chosen. The iterative
Figure 4: Control points shifted by the Culebra algorithm to avoid an obstacle.

The process of shifting control points terminates when one of three conditions is met: all segments contained within the current spline have been checked for obstacles and found to be valid segments, the algorithm has exceeded its allotted time limit, or the required shift of any given control point causes the control point to lie outside the lateral boundary of the original control points. The first case indicates that a valid path has been obtained. The generated path is then sent to the navigation computer to be used as a reference by the steering controller. In the second case, the algorithm has taken too long and whatever progress has been made in shifting the control points is sent to the navigation computer as is. The algorithm then restarts the process using the last spline sent to the navigation computer as a starting point. The reason for introducing the timeout is because the car is supposed to be traveling at a relatively high speed while the Culebra algorithm is running and the navigation computer which controls the steering needs to have updated path information at a corresponding high rate. So the timeout concept is introduced in order to ensure that the navigation computer has the most current information available. The third case can only be reached if the corridor is entirely blocked (from one side of the lateral boundary all the way to the other side) by one or more obstacles. As mentioned earlier, the Grand Challenge rules specified that at least one possible path would be available within the corridor at all times. Therefore, if the corridor is entirely blocked, an error must have occurred in the obstacle detection process. One or more of the obstacles in question must be bogus (such as a sun reflection on some surface). With no way of knowing which obstacles are false readings, the original centerline path is sent to the navigation computer along with a cautionary speed limit of 2 m/s. If indeed there is an obstacle along the centerline then the slow speed would ensure that only minor damage, if any, would be sustained by the vehicle. If, however, the fictitious roadblock is cleared successfully, then the vehicle resumes normal operation and continues driving towards its final destination. The curvature of the path was also analyzed to ensure the vehicle was able to drive it. If the radius of curvature exceeded the vehicle’s turning radius, the control points were shifted further apart from each other until the path’s radius of curvature was within the physical limits of the vehicle’s turning radius. Any segment that was moved due to this operation was checked once again for obstacles to ensure a clear, drivable path.

3.2 Speed Planning

A speed planning algorithm was used to determine the target speed at which the vehicle should travel once a valid path had been successfully generated. The algorithm uses the angle between each path segment to determine speed. A set of three sigmoid functions form the basis of the function that relates target speed and the angle between successive segments.

The shape of each sigmoid function was determined by fitting the overall function to test data obtained using a human driver. The three base functions are given by:

\[
f_1(\theta_k) = \frac{\min+(\max-\min)}{1 + e^{3(178-\theta_k)}}
\]

\[
f_2(\theta_k) = \frac{\sqrt{7}}{1 + e^{(0.05(166-\theta_k))}}
\]

\[
f_3(\theta_k) = \frac{\sqrt{7}}{1 + e^{(3(\theta_k-178))}}
\]

where \(\theta_k\) is the angle between segments \((P_{k-1}, P_k)\).
and \((P_k, P_{k+1})\), \(\min\) is the minimum target speed of 2m/s and \(\max\) is the maximum target speed of 12m/s. The three functions were then combined using equation (5) to generate the target speed for each segment of the path.

\[
f(\theta_k) = f_1(\theta_k) + f_2(\theta_k) \times f_3(\theta_k)
\]

(5)

The target speeds assigned to each control point using equation (5) were then propagated forward and backwards along the path. Backward propagation was used to ensure proper braking distances when approaching turns. Forward propagation was used to allow the LIDAR sensors to scan the road ahead when the vehicle is coming out of a turn. Figure 7 shows an example of a case in which target speed propagation is needed. Propagation was achieved by sequentially comparing the speed assigned to a control point \(P_k\) to the speed assigned to the next control point \(P_{k+1}\). This represents the target acceleration (or deceleration) for the path segment \((P_k, P_{k+1})\). If the acceleration or deceleration limits were exceeded, the assigned speed was replaced with the speed corresponding to maximum acceleration (or deceleration) depending upon each case. Figure 8 shows the same scenario after target speed propagation has been performed.

Once a target speed had been assigned to each control point, a cubic B-spline was generated using the target speeds as control points to generate a continuous target speed. The spline was then input to an integrated PD controller running on the navigation computer.

4. Discussion

The route for the DARPA Grand Challenge was over 132 miles long. Due to the nature of the terrain and the high speed required to finish the route within the 10 hour time limit, we made the assumption that most of the time the route would be relatively free of obstacles. As a result of this assumption, the path planning algorithm by default chose a path which was in the exact center of the given corridor with only a relatively infrequent need to modify that.

Since the waypoints given by DARPA were hundreds of feet apart, we decided that their waypoints were not sufficiently dense (relatively to the range of the sensors) to produce path segments. Therefore, we interpolated their waypoints to form a dense set of waypoints that were much closer together.

These dense waypoints were then used as the control points for a cubic B-spline [3] that defined the actual path of the vehicle.

We used cubic B-splines to define the actual route because B-splines produce very smooth curves which can be followed with higher vehicle speed and smoother controls. The curve resulting from a B-spline is actually a perfect fit for approximating the properties of a non-holonomic vehicle. However, a B-spline curve is
not guaranteed to pass through all of its control points, but rather it is influenced by them.

Initially, a B-spline was created that originated from the vehicle and followed the center of the corridor for 75 meters. Since this curve was in the exact center of the corridor, it offered the most clearance from tight corridor boundaries (like the 9 foot wide tunnel that was encountered during the end of the Grand Challenge). The control points for this B-spline curve were evenly spaced at a distance of around 4 meters. When the B-spline curve was evaluated, the resulting path smoothly handled greater than 90 degree turns, and could even handle 180 degree hairpin turns. Figure 8 shows the smooth path generated for a 180 degree hairpin turn.

Since we trusted that the vehicle could handle any typical off-road conditions, we did not deem it necessary to constantly try to adjust the path for the best terrain. Our strategy was that the vehicle would drive down the center of the corridor unless an obstacle was directly in the vehicle’s planned path. At this point, the path would have to be adjusted so that the vehicle avoided the obstacle.

The ease in which the B-spline curve could be altered by control point modification was the basis for the obstacle avoidance algorithm. To determine if an obstacle intersected this path, each polygon in the approximated path was checked for intersection against the obstacles in the obstacle repository. At the point where an obstacle intersected the planned path, the two control points that defined the intersected polygon were identified. These two control points then were shifted either to the left or to the right by a discrete amount and then the path was checked again to see if it intersected any obstacles. This procedure was repeated until either the planned path was free of obstacles, all possibilities were exhausted, or a set amount of time had elapsed.

Although beyond the scope of this paper, we should mention for reasons of completeness that some Reactive Behaviors were employed as well to make up the hybrid deliberative/reactive system. The three reactive behaviors that we used are the following:

- **SafetyBox Behavior** was triggered when a rectangular region directly ahead of the vehicle, and with length proportional to the vehicle’s speed, was violated by a detected obstacle. The result of the safety box being violated by an obstacle was a sudden deceleration to 2 mph. During the process of the vehicle’s deceleration, the deliberative planning algorithm would have the necessary time to plot a path that avoided the impending obstacle.

- **Corridor Blockage Behavior**. It was known that a complete blockage of the corridor would never occur, as the rules specifically stated that the corridor was passable by a standard commercial truck. Therefore, the only reasons that the corridor could be completely blocked were if a moving obstacle had temporarily blocked the corridor or if the vehicle’s sensors were registering false obstacles due to sun glare, highly reflective obstacles, and other causes. This reactive behavior first stopped the car for a set amount of time. This was to allow a temporary moving obstacle time to leave the corridor. Next, the vehicle would attempt to drive forward at a very slow pace towards the center of the corridor until the vehicle had cleared the corridor blockage. At this point, the reactive behavior would stop restricting the vehicle, and the path from the deliberative planner would once again become valid.

- **GPS Jump Behavior**. If a GPS signal jump occurred and was not properly accounted for, the vehicle might quickly attempt to regain the proper course by aggressively steering
towards the proper path. Oversteering at high speed could easily cause the vehicle to lose control or even to roll over. Therefore, we created a reactive behavior that could detect a GPS outage and then properly compensate for it. This behavior determined if a GPS jump had occurred by analyzing the distance from the center of the vehicle to the nearest point on the path. If this distance, also called the path-distance, was over a pre-determined amount (in this application it was 2 meters), the vehicle would immediately begin to decelerate. Since the vehicle could be aggressively turning at this point, special care was taken to ensure that the deceleration was accomplished with moderate braking only. As soon as the vehicle’s speed dropped below a predetermined threshold (in this application it was 3 m/s), the behavior would directly modify the vehicle’s current path by pulling the nearest control points on the path towards the vehicle. The nearest two control points were identified by finding the two control points with the shortest distance to the current location. These two control points were then pulled towards the vehicle’s current location by calculating the bearing from each of these control points towards the vehicle’s location, then projecting each control point along this bearing by the previously calculated distance. This modification would create a path that would quickly and safely bring the vehicle back onto the proper path.

5. Conclusion

The purpose of this paper was to describe the methods used in Kat-5 for performing obstacle avoidance and determining the path which Kat-5 ought to follow. There are of course other issues to this work that are outside the scope of this paper and cannot be adequately discussed within the page limitations of this paper. The major ones among these issues however, are that of actually detecting obstacles using sensory information (LIDARs) and that of actually controlling the vehicle to stay on the planned path. The first one was handled by interpreting a depth map that was produced by the LIDARs and by maintaining a geospatial representation of the findings at all times. The second was handled by using a lead-lag controller based on the classical single-track model or “bicycle model” developed by Riekert and Schunck 0.

The methodology described here is certainly useful in different path planning problems. We did have the advantage of the guarantee that one passable path would exist at all times. However, in the absence of such an assumption the method would still work if a backtracking mechanism is employed in the whole process.

Acknowledgments

This project was financed by the Gray Insurance company and this technology is further being developed by GrayMatter Inc.

C. Koutsougeras would like to dedicate this paper to Dr. C. V. Ramamoorthy.

References


