Arc Path Collision Avoidance Algorithm for Autonomous Ground Vehicles

by

Ankur Naik

Thesis Submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirement for the degree of

MASTER OF SCIENCE

IN

MECHANICAL ENGINEERING

December 15, 2005

Blacksburg, VA

Keywords: autonomous, path planning, obstacle avoidance, collision avoidance, unmanned

Arc Path Collision Avoidance Algorithm for Autonomous Ground Vehicles

Ankur Naik

Abstract

Presented in this thesis is a collision avoidance algorithm designed around an arc path model. The algorithm was designed for use on Virginia Tech robots entered in the 2003 and 2004 Intelligent Ground Vehicle Competition (IGVC) and on our 2004 entry into the DARPA Grand Challenge. The arc path model was used because of the simplicity of the calculations and because it can accurately represent the base kinematics for Ackerman or differentially steered vehicles.

Clothoid curves have been used in the past to create smooth paths with continuously varying curvature, but clothoids are computationally intensive. The circular arc algorithm proposed here is designed with simplicity and versatility in mind. It is readily adaptable to ground vehicles of any size and shape. The algorithm is also designed to run with minimal tuning. The algorithm can be used as a stand alone reactive collision avoidance algorithm in simple scenarios, but it can be better optimized for speed and safety when guided by a global path planner. A complete navigation architecture is presented as an example of how obstacle avoidance can be incorporated in the algorithm.

Table of Contents

List of Figures

Chapter 1 - Introduction

1.1 Project Overview

This thesis presents an arc (curvature) based collision avoidance algorithm that was developed for Virginia Techs entries into the 2003 and 2004 Intelligent Ground Vehicle Competition (IGVC). The same algorithm was also used on our 2004 DARPA Grand Challenge entry, Cliff. The IGVC vehicles are required to avoid construction barrels, white and yellow lines painted on grass and simulated potholes (white circle painted on grass). The DARPA Grand Challenge Entry 'Cliff' is required to navigate 3D terrain as well as negotiate any kind of obstacle encountered. All of the IGVC vehicles use differential drive systems. Previous IGVC entries have used the Vector Field Histogram for obstacle avoidance. The Vector Field Histogram requires tuning of a PID (Proportional Integral Differential) loop in order to balance path accuracy versus smoothness. This has proven problematic in the past. Cliff cannot use the Vector Field Histogram because it is Ackerman steered and is therefore incapable of following vector paths. These are reasons why a new arc based method was developed. To date, the arc-based collision avoidance algorithm has only been proven to run on low-speed vehicles. At higher speeds, where dynamic effects become significant, the clothoid-based algorithms may be necessary to account for the finite rate of change of path curvature.

1.2 Motivation

This research was motivated by the need for a simple obstacle avoidance algorithm that could be implemented on a variety of autonomous ground vehicle platforms being developed at Virginia Tech. In particular, the vehicles developed for the

Intelligent Ground Vehicle Competition (IGVC) and the DARPA Grand Challenge were targeted for implementation of the algorithm. These vehicles are shown in figures 1.1 and 1.2. An ideal collision avoidance algorithm would be interchangeable between robots with minimal need for tuning. Collision avoidance is required to react to obstacles detected by the sensors that were not anticipated ahead of time. Challenges include reacting to obstacles, reaching the specified target and avoiding dead ends and local minima.

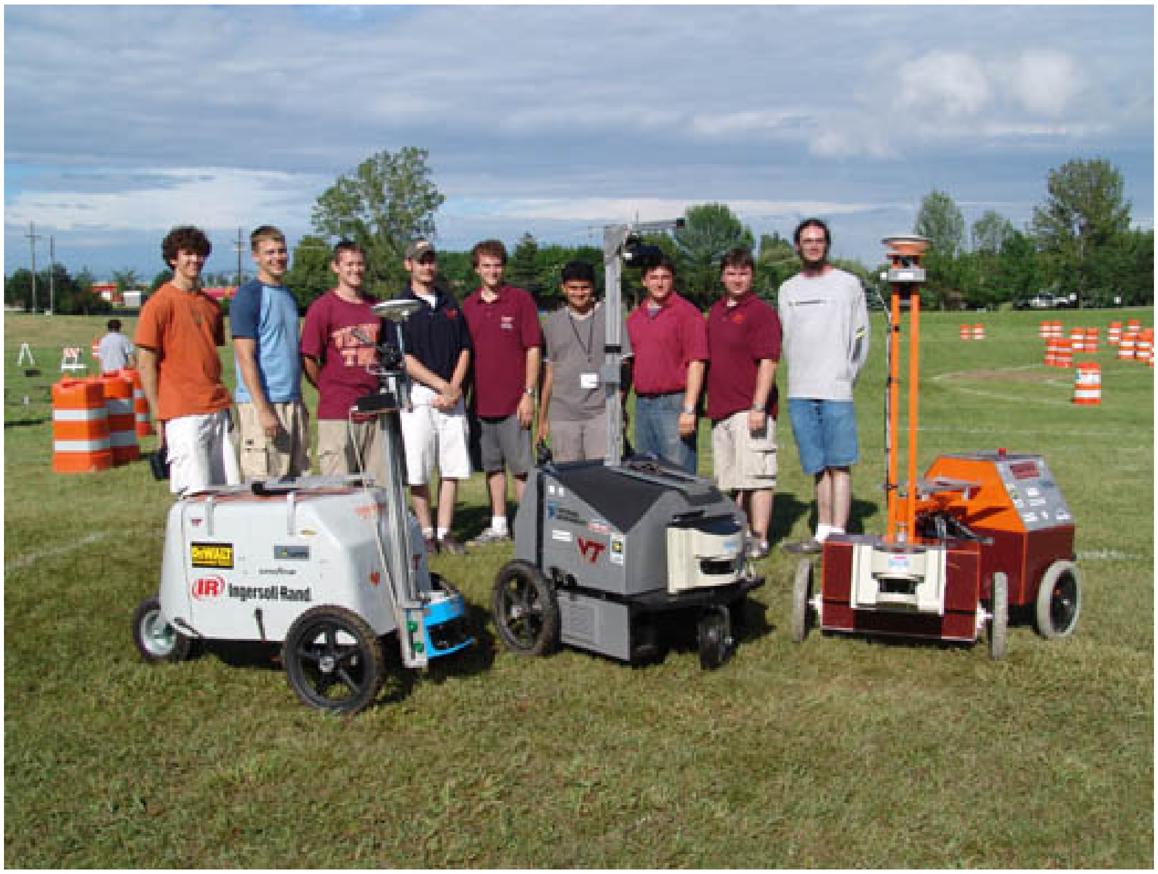

Figure 1.1- IGVC Entries Optimus, Johnny5 and Gemini with the Virginia Tech Autonomous Vehicle Team 2004(AVT 2004)

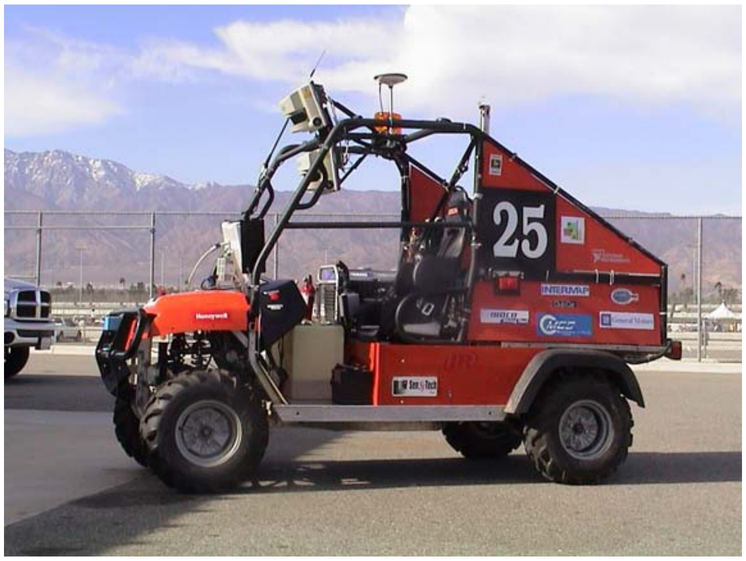

Figure 1.2-2004 DARPA Grand Challenge Entry, Cliff

Chapter 2- Review of Literature

The development of collision avoidance and path planning algorithms has yielded several well known approaches. These algorithms can be separated based on how they model the path and by how they handle kinematic and dynamic effects of the real vehicle.

2.1 Shortest Path Algorithms

Shortest path algorithms like Dijkstra, grassfire, and A* will find an optimal path between two points on a map. Unfortunately, these paths do not take vehicle motion (kinematic) constraints into account. As a result the path is only guaranteed to be traversable by holomonic vehicles (vehicles that can move in any direction with an independently controlled orientation). Such "omni-directional" vehicles are typically not feasible for outdoor navigation in rugged terrain. These algorithms are usually dubbed path planning algorithms because they work better at longer ranges but can rarely be used for short range collision avoidance. Of the shortest path algorithms, the A* and D* (dynamic A*) are the most popular because they use heuristics to reduce the computation time significantly. The A* algorithm works by first converting a map to a graph. That is each cell on the map becomes a node with edges connecting the cells to their neighbors. For a square grid map the cells can either be 4 connected allowing only vertical and horizontal motion or 8 connected allowing diagonal motion. Each edge has a cost representing the difficulty of crossing it. This is usually the length of the edge plus a factor representing the difficulty of the terrain (usually ∞ for impassible obstacles). The algorithm works by first going to the starting node. The plausibility function

$$f^*(n)=g(n)+h^*(n) \tag{2.1.1}$$

is then calculated for each adjacent node. The term g(n) is the sum of costs of all the edges crossed to reach the node and h*(n) is the estimated cost of reaching the goal node. The estimate h*(n) must always be equal or less than the actual value otherwise good paths may be eliminated. For this example h*(n) will be the straight line (Euclidian) distance to the goal node, which is always lower then the actual cost. Once f*(n) is calculated for the adjacent nodes the node with the lowest f* is expanded. That is f* is calculated for each of its adjacent nodes. The node with the lowest f* (including previously unexpanded nodes) is then expanded. This process is continued until the goal node is reached and no other unexpanded node has an f* with a lower value then the goal node. Figure 2.1 shows a worked example.

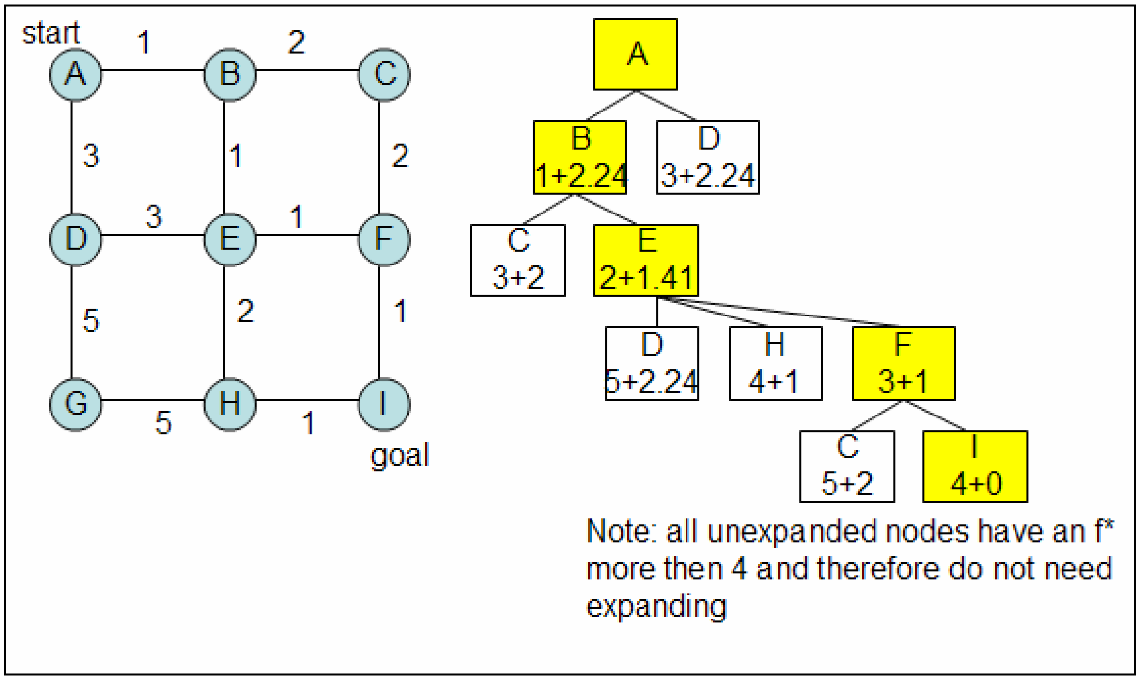

Figure 2.1- Worked A* Example

The A* algorithm can be used for short range collision avoidance only at very low speeds due to the high computation time. When used on a square grid map, the algorithms are typically useful only for holomonic or differential drive vehicles that can traverse any path, but this must be done at low speeds due to the continuous sharp turns. The sharp turns present because on a square grid map the motions are limited to straight lines and 90 degree turns. This is illustrated in figure 2.2. Instead shortest path algorithms are better suited for long range path planning. The output of a shortest path algorithm can be used to guide a more vehicle friendly short range collision avoidance algorithm. Shortest path algorithms in general are computationally expensive and therefore have not been feasible in the past, but with today's faster processors, they are beginning to regain popularity in the path planning/ collision avoidance arena.

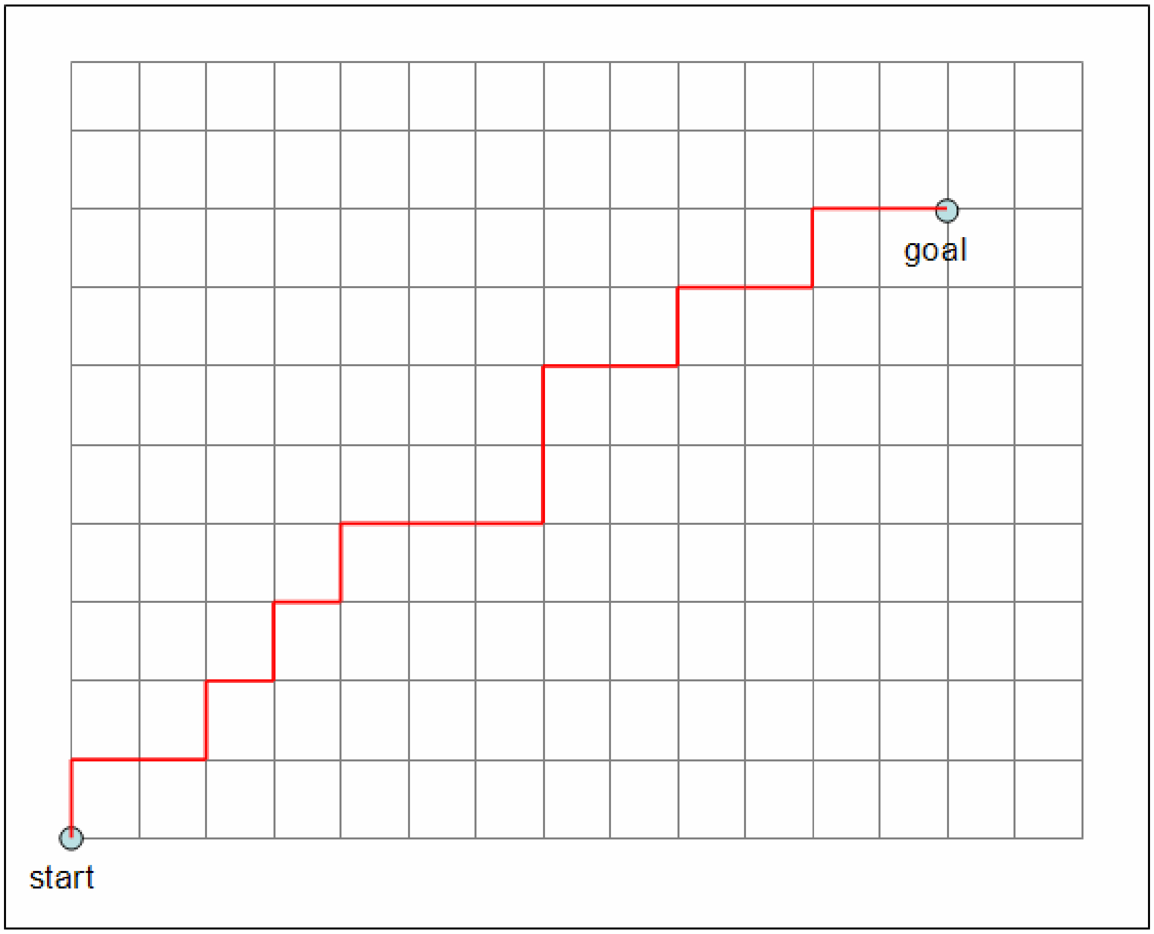

Figure 2.2- Shortest path on a square grid map

2.2 Vector Based Methods

Vector based methods, such as the Vector Field Histogram (VFH), typically use vector or straight lines as the basic path model. The Vector Field Histogram was developed by J. Borenstein and Y.Koren [1]. The VFH method starts with a certainty grid which is a global map (map is stationary relative to the world) with cell values representing how certain the sensors are that an obstacle is present in the region represented by those cells. This is done by incrementing the cell values every time a sensor detects an obstacle cells location. This assumes that the more times an obstacle is detected, the higher the chance that is exists. The cells that are in the immediate vicinity (usually within 1 to 5 meters) of the vehicle are taken and converted to a local map (relative to the vehicle) with the vehicle at the center (xo,yo=(0,0)) facing up. The vector field histogram is a histogram representing the probability that obstacles exist in each direction. The histogram is produced first by calculating the angle (β) and distance (d) each cell (i, j) is relative to the vehicle using the following equations.

$$\beta(i,j)=\text{atan}((yi-yo)/(xi-xo)) \quad (2.2.1)$$

$$d(i,j)=\text{sqrt}((yi-yo)^2+(xi-xo)^2) \quad (2.2.2)$$

The cells are separated by their angle β into 1 degree sets from 0 to 360 degrees. The histogram value (H) for each set (k) is calculated using the following equations

$$H(k)=\Sigma\ m(i,j) \quad (2.2.3)$$

$$m(i,j)=c(i,j)^2(a-bd(i,j)) \quad (2.2.4)$$

where c(i,j) is the intensity and a and b are constants. The histogram should something like figure 2.3 taken directly from the text [1]

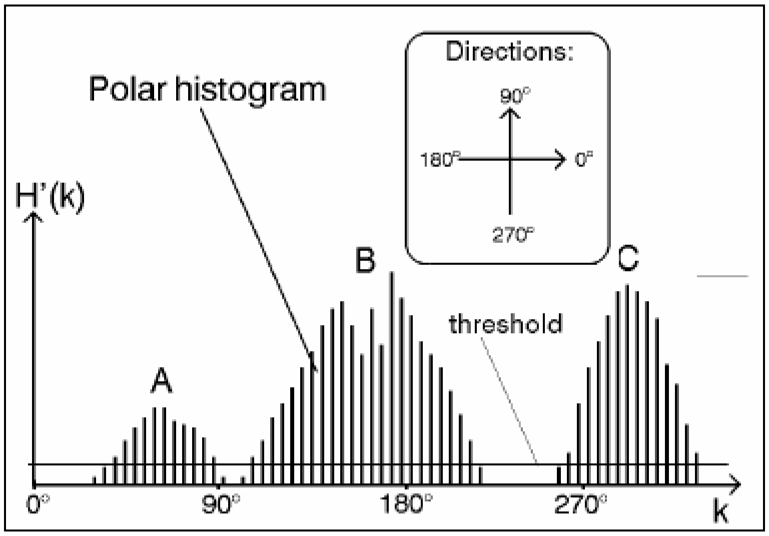

Figure 2.3- Vector Field Histogram [1]

The histogram is thresheld to remove small or insignificant H(k). The vehicle then navigates towards gaps in the histogram that are closest to the target direction. The vector field histogram outputs a desired change in heading (Δθ) which must then be converted to a translational velocity (tv) and rotational velocity (rv) for differential drive vehicles. It is impossible for the vehicle to stop and turn every time a change in heading is required, due to the acceleration limits of the vehicle, in order in order to exactly follow the vector path. It is more desirable for the vehicle to remain in constant forward motion and only slow down when sharp turns are required. Virginia Tech previously used PID loops with Δθ as an input in order to control tv and rv in order to get smoother motion. Path accuracy versus motion smoothness has been a continuing problem with Virginia Tech's use of the Vector Field Histogram. Since Ackerman steered vehicles cannot steer vector paths, the vector field histogram is not typically suitable for such vehicles. The Vector Field Histogram only uses local data and therefore it can easily get stuck in dead ends or local minima. Also, the Vector Field Histogram is not designed as a long range path planner but more as a short range collision avoidance algorithm. One advantage of the Vector Field Histogram is that it is computationally the fastest of the algorithms discussed in this thesis.

2.3 Curvature Based Methods

A well known curvature or arc-based method is the Curvature Velocity Method (CVM) developed by Reid Simmons of Carnegie Mellon [6]. The CVM works by converting the XY obstacle map into a 3D configuration space where the 3 dimensions are rotational velocity (rv), translational velocity (tv) and curved distance to obstacle (dc). This is illustrated in figure 2.4 taken directly from the text [6].

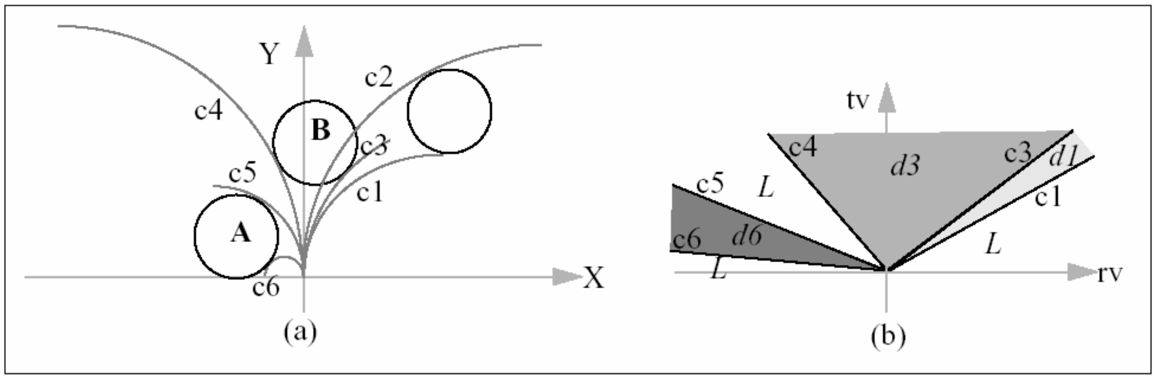

Figure 2.4- CVM Configuration Space [6]

The curvature velocity method assumes that all obstacles are circles. For each obstacle the range of curvatures (c) that causes interference with the obstacle is calculated. This is shown in figure 2.5 taken directly from the text [6].

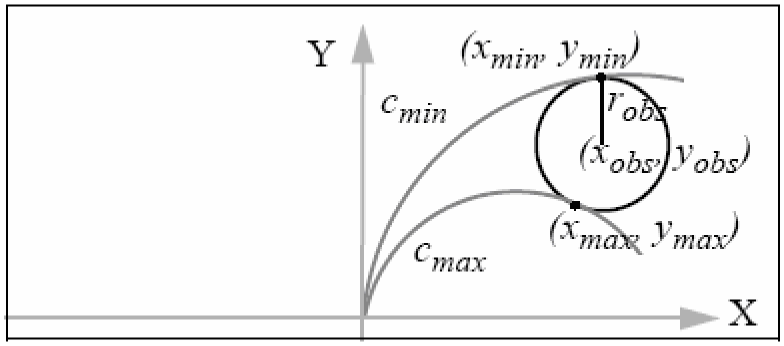

Figure 2.5- Min and Max Curvature blocked by an object [6]

c_{min}, c_{max}, x_{min}, y_{min}, x_{max} and y_{max} are calculated using the following equations.

$$c_{\min} = 2(x_{obs} - y_{obs}) / \sqrt{x_{obs}^2 + y_{obs}^2 + r_{obs}^2} \tag{2.3.1}$$

$$c_{\max} = 2(x_{obs} + y_{obs}) / \sqrt{x_{obs}^2 + y_{obs}^2 + r_{obs}^2} \tag{2.3.2}$$

$$x_{\min} = (x_{obs} - r_{obs}) / (1 - (c_{\min} \cdot r_{obs})) \tag{2.3.3}$$

$$y_{\min} = y_{obs} / (1 - (c_{\min} \cdot r_{obs})) \tag{2.3.4}$$

$$x_{\max} = (x_{obs} + y_{obs}) / (1 + (c_{\max} \cdot r_{obs})) \tag{2.3.5}$$

$$y_{\max} = y_{obs} / (1 + (c_{\max} \cdot r_{obs})) \tag{2.3.6}$$

For each intersection point (x_{min},y_{min}) and (x_{max}, y_{max}) the curved distance to the obstacle (d_c) is calculated. How this is done is shown in Figure 2.6 taken directly from the text [6] and the following equations.

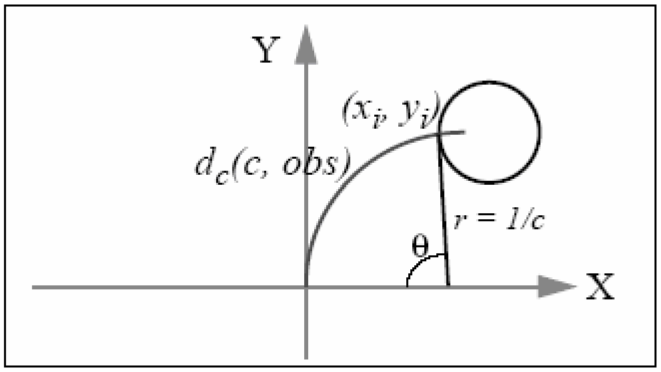

Figure 2.6-Calculating the Curved Distance to Obstacle [6]

$$\theta = \begin{cases} a\tan(y_i/(x_i - 1/c)) & c < 0 \\ \pi - a\tan(y_i/(x_i - 1/c)) & c > 0 \end{cases} \tag{2.3.7}$$

$$d_c(c, obs) = \begin{cases} y_i & c = 0 \\ |1/c| \cdot \theta & c \neq 0 \end{cases} \tag{2.3.8}$$

The lower d_c of the two is taken as the d_c for the entire range from c_{min} to $c_{max.}$ If d_c is less then a certain length L (3 meters) then the object is rejected as being too far away otherwise it is plotted on the configuration space map as shown in figure 2.4 where dc is the axis coming out of the page. Curvatures (c) are converted to a range of tv, rv combinations using the equation

$$c=rv/tv \tag{2.3.9}$$

The best combination of tv and rv is found by passing the data through an objective function that favors higher forward speeds, higher distances to obstacles, and rotational velocities that place the vehicle closest to the desired heading after a fixed time constant (Tc). This and other curvature based methods have only been used for low-speed navigation. Usually clothoid-based approaches are used when higher speed is required in order to account for vehicle dynamics. The CVM method is computationally fast, but not quite as fast as the vector field histogram due to its more complex calculations. The CVM is only suited for short range collision avoidance. Without the guidance of a long range path planner such as the A* algorithm, the vehicle can get trapped in local minima or dead ends

2.4 Clothoid Based Methods

Clothoids paths are simply segments of the Eulers spiral, meaning the path segments have a constant change in curvature. Clothoids are curves that have a curvature that varies linearly with the distance. Clothoid paths are usually calculated by numerically simulating a vehicle in constant forward motion and with a constantly changing steering angle. This is done because there is no easy formula to generate a clothoid path. Shown in Appendix A is a Matlab script programmed by Andrew Bacha which generates a clothoid for Ackerman steered vehicles. For more general equations for clothoid segments, I will present the equations for a clothoid taken directly from the text [9].

$$c(s) = k\, s + c_i \tag{2.4.1}$$

where s is the distance along the curve, c(s) is the curvature, k is the rate of change of curvature (sharpness) and c_i is the initial curvature. The heading θ(s) is given by the equation

$$\theta(s) = \theta_i + \int_0^s c(\zeta)d\zeta = \frac{k}{2}s^2 + c_i s + \theta_i \tag{2.4.2}$$

where θ_i is the initial heading. The x and y location is given by the following equations.

$$x(s) = x_i + \int_0^s \cos\theta(\zeta)d\zeta = \int_s^0 \cos(\frac{k}{2}\zeta^2 + c_i\zeta + \theta_i)d\zeta + x_i \tag{2.4.3}$$

$$y(s) = y_i + \int_0^s \sin\theta(\zeta)d\zeta = \int_s^0 \sin(\frac{k}{2}\zeta^2 + c_i\zeta + \theta_i)d\zeta + y_i \tag{2.4.4}$$

Where x_i, y_i are the initial x, y location. The integrals in equations 2.4.3 and 2.4.4 are usually solved numerically. Clothoids are used by the National Institute of Standards and technology (NIST) for collision avoidance on their robotic Humvee [5]. The algorithm is called the NIST Real time control system (RCS). In the NIST RCS application, the clothoids are pre-computed specifically for the humvee and stored in a look up table. The clothoids that correspond to the current speed and steering of the vehicle are used as the first segments in a radial map. This is shown in figure 2.7 taken directly from the text [5]. An A* algorithm is then run on the radial map to select the best clothoid segment. This process is repeated continuously so that a new clothoid path is selected before then end of the current clothoid path is reached. This algorithm is computationally expensive because each path segment must be checked to see if it is clear of obstacles. After this is done, the A* algorithm, which is also computationally expensive, must be run. This algorithm does well at long range path planning as well as short range obstacle avoidance. It has been shown in field testing to run reliably up to 35mph, which is the fastest of the algorithms describes in this paper

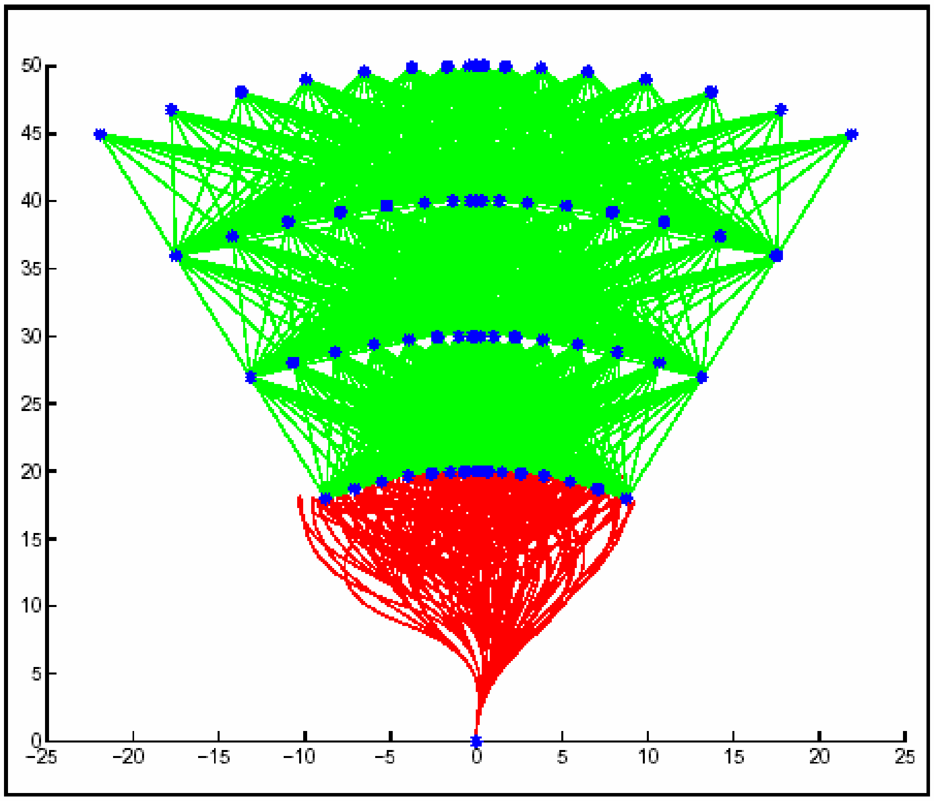

Figure 2.7- NIST RCS Radial Map with Clothoid Path Segments [5]

2.5 Dynamic Window Approaches

Dynamic window approaches take any algorithm from any path model, but only the paths achievable in the next cycle can be selected. This assures that nothing beyond the physical ability of the vehicle can be commanded. Because of the narrowed search area, the algorithm becomes much more susceptible to local minima or dead ends. This approach is best when you want the robot to drive less aggressively and it is guided by a higher-order path planning algorithm.

Chapter 3 - Algorithm and Comparison and Discussion

3.1-Algorithm Comparison

Most of the algorithms work extremely well in controlled environments or with extensive fine tuning. Figure 3.1 shows how I categorize each algorithm by its path planning ability versus its collision avoidance ability. The abscissa qualitatively represents how well the algorithm performs at path planning, while the ordinate axis comparatively shows its effectiveness at collision avoidance.

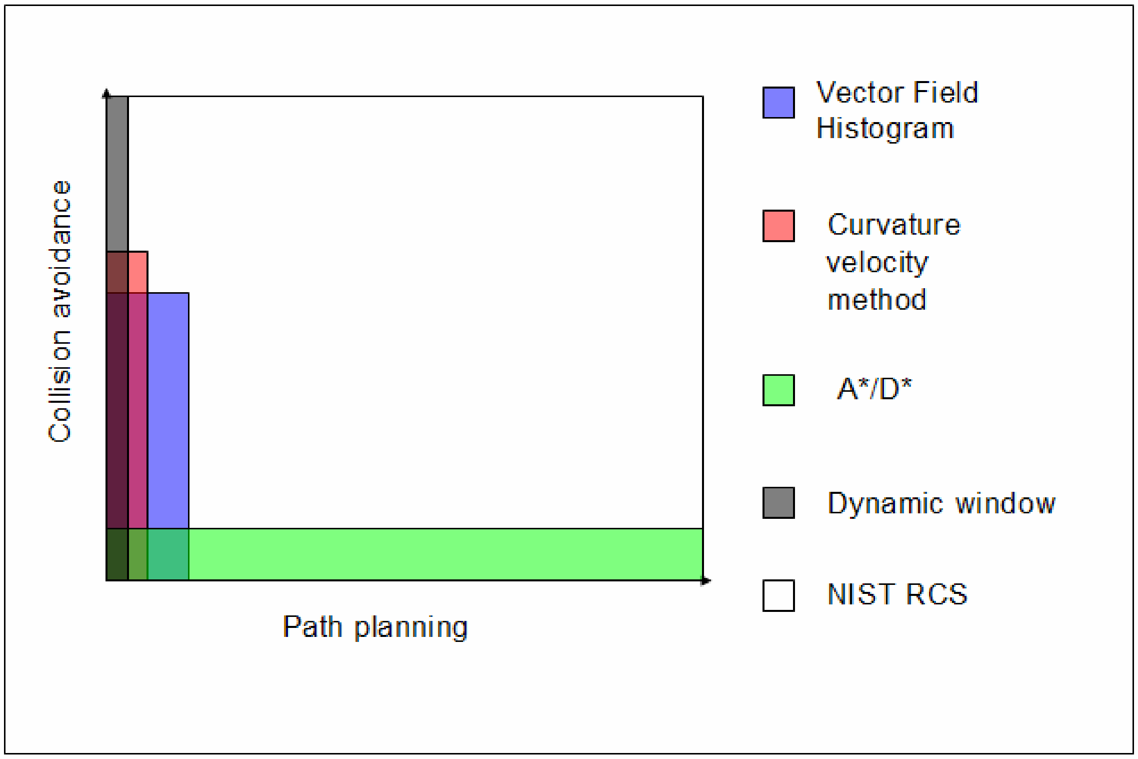

Figure 3.1- Algorithm Classification Chart

Path planning refers to the algorithms ability to plan a path that will keep the vehicle out of dead ends and assure progress towards the goal. Collision avoidance refers to the algorithms ability to avoid immediate obstacles while the vehicle is in motion.

Path planning is generally better for algorithms that can be used over longer ranges. The Shortest Path Algorithms such as the A*/D* algorithms generate piecewise paths that can be generated to maneuver around any combination of obstacles and therefore can be used for an infinite range limited only by sensor range.

Collision avoidance ability is generally better when using algorithms that account for vehicle kinematics and dynamics. The shortest path algorithms do not account for either, and so they generally create paths that are full of angular discontinuities (infinitely sharp turns) that are generally not feasible by vehicles that are have continuous forward velocity and non-zero turn radius.

The vector field histogram (VFH) is better then the curvature velocity method (CVM) for path planning because in the absence of obstacles, a single vector leading to the goal is already the most efficient path to the goal and the VFH method only deviates from this path when obstacles are present. This is not true for the CVM method which uses arc paths which loop back towards the vehicle and therefore cannot be used over long ranges. Both algorithms are prone to dead ends and local minima.

The CVM method, however, is better at collision avoidance then the VFH method because the arc paths take vehicle kinematics into account. Vehicle kinematics and dynamics affect path following accuracy at higher speeds. This means that vehicles will more closely follow the arc path generated by the CVM then the vector path generated by the VFH and therefore is more likely to avoid the obstacle. The CVM method however only allows for binary obstacles.

I categorize the NIST RCS algorithm as covering the entire chart because is considers both long range path planning using an A* algorithm and short range collision avoidance with consideration for vehicle kinematics and dynamics using clothoids. Unfortunately the NIST RCS algorithm is computationally intensive.

I categorize the dynamic window approaches high on collision avoidance because it takes vehicle dynamics into accounts but it ranked lowest on the path planning due to its narrow search range which leads to its inability to avoid local minima.

3.2-Local versus Global Path Planning versus Collision Avoidance

The primary difference between a local and a global path planner is that for a local path planner, the path is selected based on current sensor data, in the vehicle co-ordinate frame, whereas the global path planner works on a ground referenced 'global' map covering the entire terrain and plans a path from the vehicles current location to the goal.

In general the path generated by a local path planner is only valid up to the vehicles sensor range and will be re-planned continuously as the vehicle moves and new sensor data is acquired.

The path generated by a global path planner is valid from the vehicles current location up to the goal. The path only needs re-planning when the vehicle deviates from the path or new obstacles are detected which occlude the path.

The use of a global path planner requires that the terrain the vehicle will traverse is known in advance. For the Autonomous Challenge component of the IGVC nothing is known about the terrain in advance. The only known rule is that that course is bounded by two white lines [7]. The path planner in this case simply navigates to a point that is within the vehicles sensor range that lies between the two lines. This point continually moves ahead with the vehicle and thus the path is continually changing. The algorithm then uses a collision avoidance algorithm the avoid obstacles. This is an example of a true local path planner.

For the waypoint challenges such as the DARPA Grand Challenge and the IGVC navigation challenge, only the waypoints the vehicle must reach are known. There is no information about the terrain or the obstacles. Therefore the approach used for path planning is simply to steer towards the next waypoint. This could be considered a global path planner because the path is valid beyond the sensor range of the vehicle, however there is no accounting for the obstacles and terrain that lie beyond the sensor range of the vehicle. Instead the algorithm relies on collision avoidance to navigate around obstacles.

Collision avoidance algorithms, on the other hand, are only concerned with the immediate motion of the vehicle. That is, only a single motion is commanded and this motion is updated as the vehicle moves. The algorithms presented in chapter 2, with the exception on the shortest path algorithms and the NIST RCS, project the paths for each motion command to see if they are free of obstacles and a clear path is selected based on input from the path planner. This however does not mean that the projected path will be followed through to the end. The motion command will be updated and the vehicle progresses forward and new sensor data is acquired.

The purpose of the algorithm presented in this paper is purely for collision avoidance. The path planning algorithms presented for each challenge being faced by the Virginia Tech robots were independently tested using test scenarios free of obstacles. These path planning algorithms worked sufficiently.

3.3 Path Model Comparison

The three path models to be compared are Vector paths used by the VFH method, Arc paths used by the CVM and Clothoid Paths used by the NIST RCS algorithm.

The vector paths are defined by a heading (θ). When commanding a change in heading, there is a tangent mismatch between the path the vehicle is currently on and the path the vehicle is being commanded to take. This means there is a first order discontinuity in the path of the vehicle. For a differentially driven vehicle the lowest order motion command acceptable is a rotational velocity (rv or θ') and a translational velocity (tv). There is no way to command Δθ directly. This means the Δθ must be passed to a control loop (usually PID) which then commands a sequence of rotational and translational velocities (rv and tv) which will result in the change of heading. The tangent mismatch cannot be traversed accurately unless the vehicle is already at a complete stop. The PID loop parameters will determine how sharply the vehicle reacts to achieve the change in heading. The final result is a smooth but inaccurate path as opposed to a sharp change in heading. This is illustrated in figure 3.2.

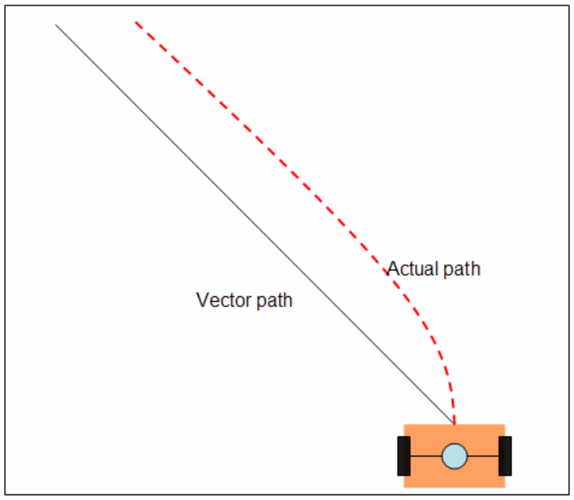

Figure 3.2- Smoothing of Tangent Mismatch

This effect worsens as forward speeds increase. It is for this reason that vector paths cannot be steered accurately. This in turn affects the collision avoidance performance because the actual path does not follow the commanded path accurately. Vector paths also cannot be steered by Ackerman steered vehicles. Instead Ackerman steered vehicles approximate the vector path, which results in a similar smoothing effect as seen in figure 3.2.

Arc paths are defined by a turn radius or a constant tv and rv. When a change in arc path is commanded, the tangents between the current path and the commanded path match up, unlike the vector paths. However, there is a curvature mismatch. That means there is a second order discontinuity in the path of the vehicle. For both a differentially driven vehicle and an Ackerman steered vehicle, a turn radius can be commanded directly. That is, for a differentially driven vehicle, a radius of curvature can be converted to wheel speeds. For an Ackerman steered vehicle a radius of curvature can be converted directly to a steering position. This however does not mean that a change in turn radius can be achieved instantaneously. How fast the vehicle responds to the change in command depends on the dynamics of the actuators and is therefore a function of the particular vehicle, steering actuators and controllers.

Clothoid paths are defined by an initial curvature, a constant rate of change of curvature and a constant forward speed (tv). The initial curvature is matched with the previous path. The only changes commanded are the forward speed and the rate of change of curvature. This means that when a new clothoid is commanded, both the path tangents and the curvatures match up. This means the discontinuity in the path is moved up to the third order. For differentially driven vehicles, a change in clothoid would be commanded as wheel accelerations as opposed to wheel speeds for arc paths. For an Ackerman steered vehicle the change in clothoid would be commanded as a vehicle speed and a steering motor speed as opposed to a vehicle speed and steering motor position for an arc path. For Ackerman steered vehicles, clothoid paths do not account for vehicle acceleration or steering motor acceleration. For differential drive vehicles, clothoid paths do not account for change in wheel acceleration or jerk.

Clothoid paths, however are difficult to generate. They are usually created using a numerical simulation of the robot motion (See Appendix A). Once the path is generated, it must be checked point by point to see if it is clear of obstacles. This is because the integrals in equations 2.4.3 and 2.4.4 must be solved numerically to determine the path. At this point, the x and y location of an obstacle can be used to see how close the obstacle lies to the clothoid.

Previous IGVC teams have successfully used the VFH method, which uses vector paths. The Arc or constant curvature paths are better then vector paths, because they match the vehicle's current path tangent. This is indicated by the performance of the CVM. Arc paths can also be used on Ackerman steered vehicles. Clothoid paths, though technically superior to arc paths, increase the complexity of the calculations significantly. For the speeds required by the IGVC and the 2004 DARPA Grand Challenge Qualifier, testing showed that Arc based paths would be sufficient. However, it was not possible to simply use the CVM. This is because the CVM only works on binary obstacles. The DARPA Grand Challenge required that the vehicle navigate through varying terrain with obstacles that had a varying degree of traversability. Also the CVM method would be too computationally expensive for the number of obstacles being considered.

Chapter 4- Path and Vehicle Model

When implementing a collision avoidance algorithm, important considerations include what path model to use and what vehicle model to use. For the reasons discussed in Chapter 3, I chose to use Arc (constant curvature) paths as the path model for this collision avoidance algorithm

It is also important that the geometry of the vehicle be accurately modeled. The simplest vehicle models are points and circles. While these are computationally simple for collision detection calculations, they are often too inaccurate to be practical. The Arc based model allows the vehicle geometry to be included without much additional complexity. The geometric model used to represent the vehicle depends on the vehicle in question. Most real vehicles are rectangular, so a rectangular model is presented.

4.1 Arc Path Model

The arc or constant-curvature path model is simply a segment of a circle. It is defined by a constant radius, a starting and ending point, and a center of curvature. Figure 4.1 shows how the arc model applies to differential drive vehicles. For use in this algorithm, the arc path can be defined by a single turn radius. A right turn has a positive radius and a left turn has a negative radius. A straightforward path has an infinite radius and therefore must be treated as a special case. The path starts at the vehicle's current location with the tangent pointing straight ahead. This means the center of curvature lies on the x-axis with an x value equal to the turn radius, therefore the center of curvature does not need to be defined separate of the turn radius. The arc is assumed to go on infinitely. This is later limited by the range at which obstacles are considered

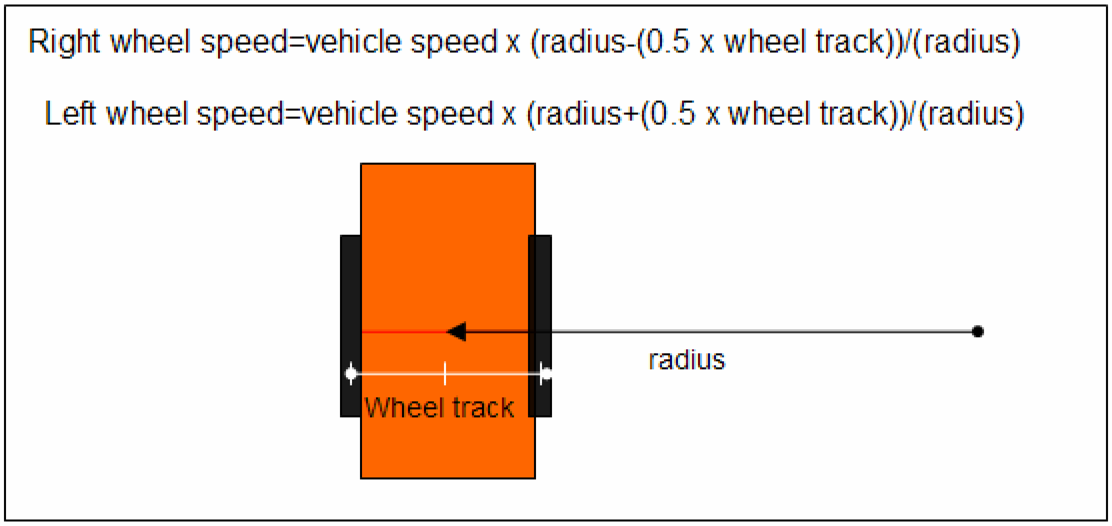
Right wheel speed=vehicle speed x (radius-(0.5 x wheel track))/(radius)
Left wheel speed=vehicle speed x (radius+(0.5 x wheel track))/(radius)
radius
Wheel track

Figure 4.1- Arc Path Mathematics for Differential Drive Vehicles

Most commercially available passenger vehicles use a mechanical linkage-based Ackerman steering mechanism. Figure 4.2 shows how the arc based path is applied to Ackerman steered vehicles. Again the Arc path can be defined as a single radius for the same reasons described earlier. The turn radius can be calculated using the outer wheel angle. The steering angle of a vehicle is defined by the Society of Automotive Engineers (SAE) as the angle taken by the outer wheel i.e. right wheel for left turns and left wheel for right turns. This is done because, in an Ackerman steered vehicle, the steered wheels do not rotate the amount in a turn. The Arc Path steering angle as shown in figure 4.2 is the angle a virtual wheel in between the front two wheels would take in order to steer the along a turn radius. This is the angle commonly used in the in the obstacle avoidance arena and it is based on the bicycle model of a vehicle. Using the Arc path steering angle simplifies the equations of motion as they do not need to be defined separately for right and left turns. It could be assumed that steering motor input is proportional to the steering angle but this is actually a non-linear relationship. The best approach is to measure the steering motor input for each steering angle and use a look up table in software.

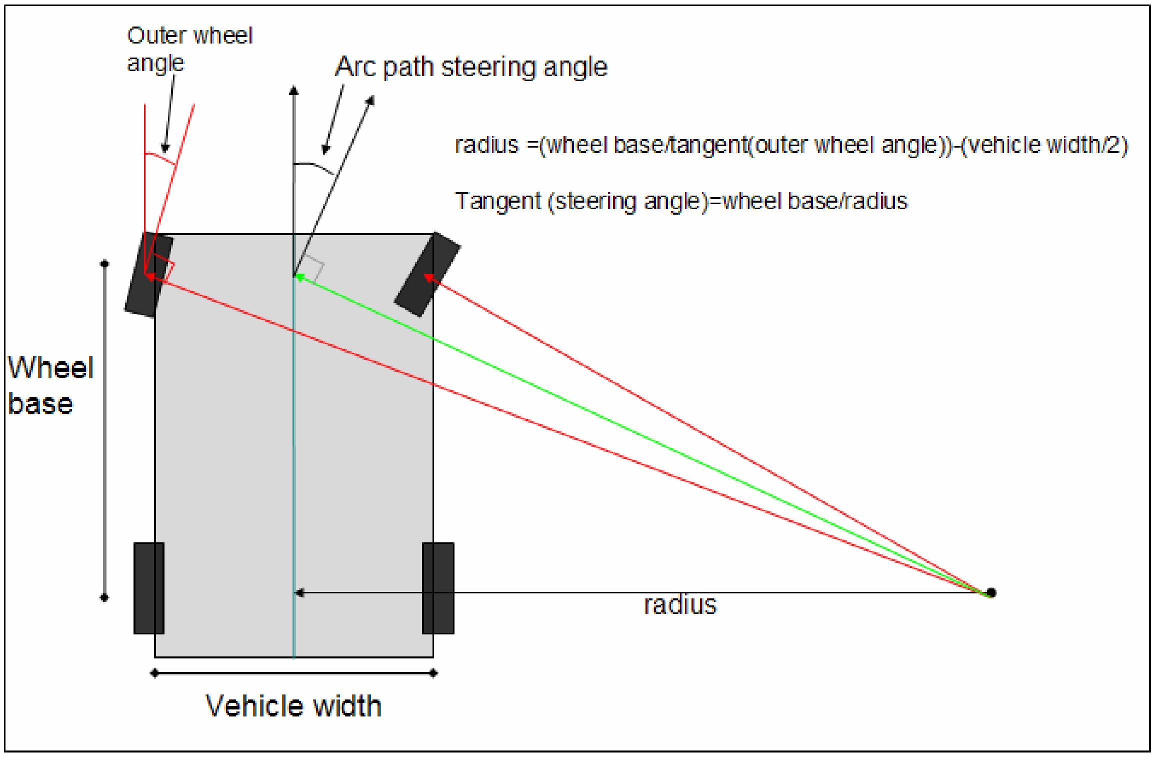
Outer wheel angle
Arc path steering angle
radius =(wheel base/tangent(outer wheel angle))-(vehicle width/2)
Tangent (steering angle)=wheel base/radius
Wheel base
radius
Vehicle width

Figure 4.2- Arc Paths with Ackerman Steered Vehicles

4.2- Vehicle Model

Since most vehicles are generally rectangular in shape, using a circumscribed circle as the vehicle model will require more clearance to pass obstacles then if a rectangular vehicle model is used. This is illustrated in figure 4.3. On the Grand Challenge Vehicle ‘Cliff’, the resulting clearance required was 5 feet for the rectangular model versus 9ft for the circular model. Using arc paths allows the use of a rectangular vehicle model because the inner and outer radius covered by a single path can be computed easily. This is illustrated in figure 4.4. Given that only a finite number of arc paths are used in the algorithm, it is possible to pre-compute the inner and outer radii for any geometry and store them for later use, reserving valuable precious processing time for obstacle avoidance.

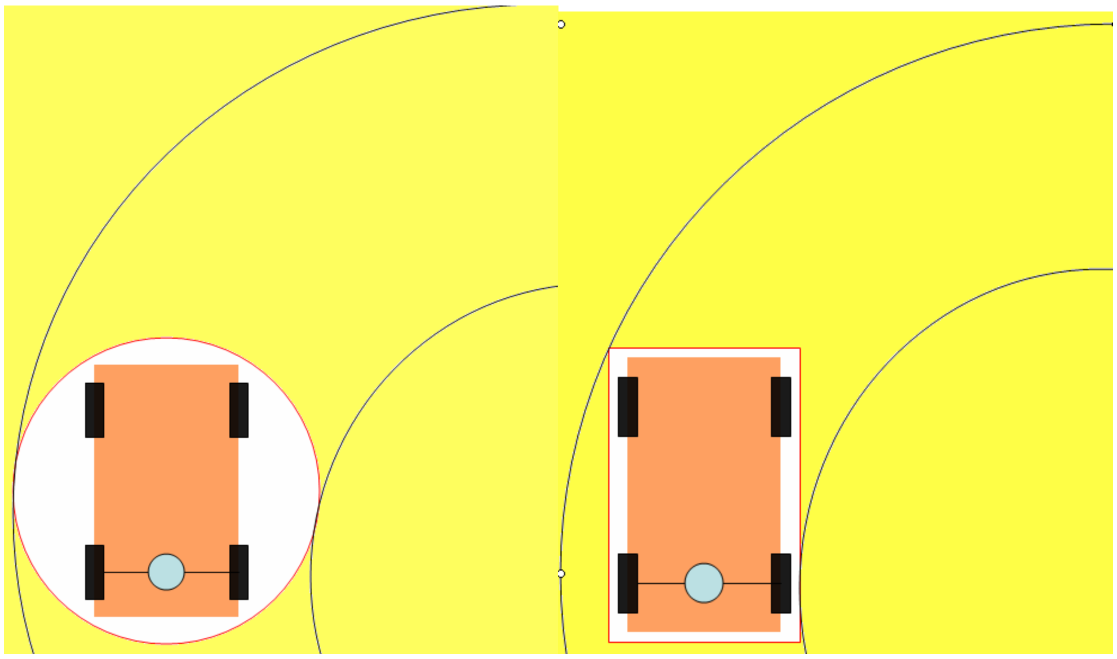
Figure 4.3- Circular Vehicle Model versus Rectangular Vehicle Model

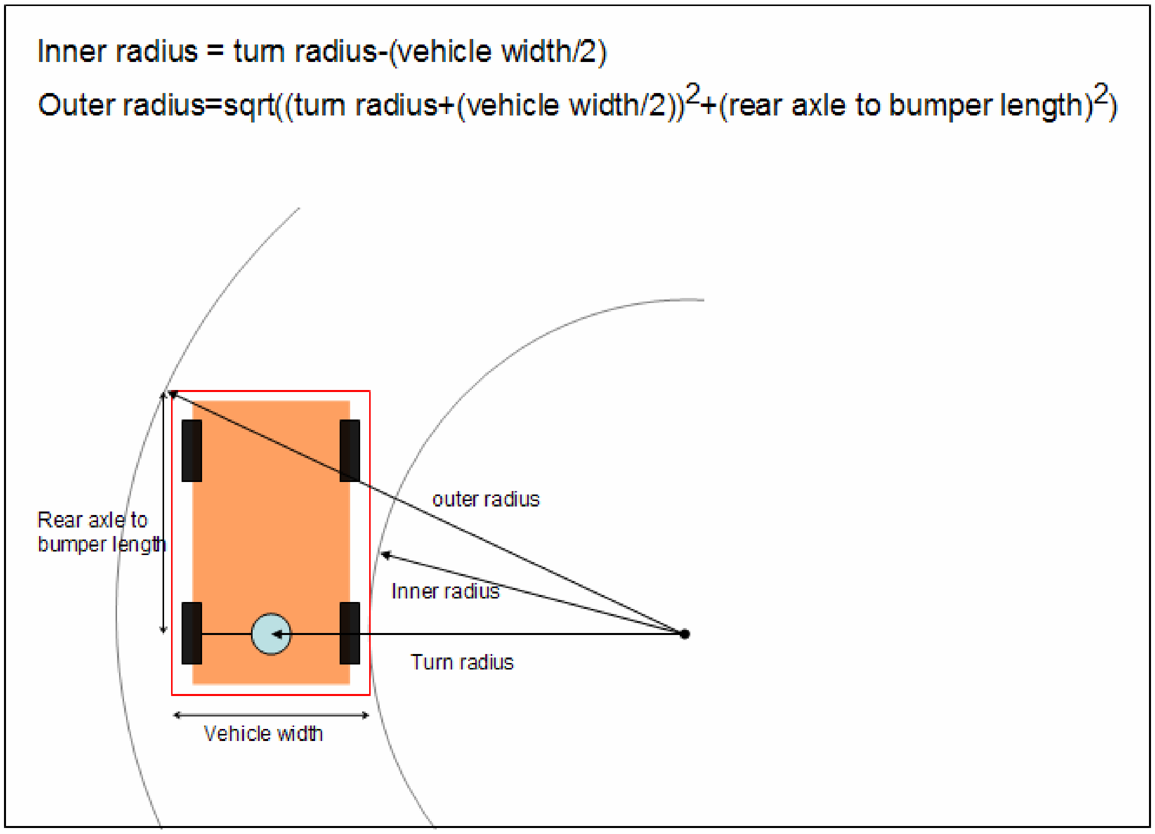
Inner radius = turn radius-(vehicle width/2)
Outer radius=sqrt((turn radius+(vehicle width/2))^2+(rear axle to bumper length)^2)
outer radius
Rear axle to bumper length
Inner radius
Turn radius
Vehicle width

Figure 4.4- Inner and Outer Radius Calculation for a Rectangular Vehicle Model

Chapter 5- Sensor Fusion and Path Selection

5.1-Sensor Data Processing

This algorithm is designed to work with any sensor that can give the XY location of an obstacle in front of the vehicle. Both the DARPA Grand Challenge and IGVC vehicles use a forward facing singe plane laser range finder to locate obstacles. The laser range finders we use are produced by SICK Optics. The laser range finder measures the distance to an obstacle by measuring the time it takes for the laser signal to travel from the range finder to the obstacle and back. The laser range finders can measure a 180 degree plane in 0.5 degree increments at up to 75 times per second. It has a range of up to 80 meters.

Cliff has two laser range finders pointed downwards (see figure 1.2), which combined with the data from an inertial navigation unit, maps the 3D terrain in front of the vehicle. The 3D data is stored as a displacement map which is an array of heights with 25cm spacing. The map is then passed through a 3x3 Laplacian edge detection filter to find changes in gradient.

The IGVC vehicles have a camera mounted on a 6 ft mast pointing downwards at a 45 degree angle. The cameras are used to detect the white lines and simulated potholes on the Autonomous challenge. This is done by splitting the images vertically, then for each half, taking the brightest pixel on each row. These pixels are then corrected for perspective distortion. Next, the pixels are passed through a Hough line transform which fits a line to the points on each side of the vehicle for use by the global path planner. The pixels are also plotted on an occupancy grid which is a matrix representing equivalently spaced points on a square grid. Matrix values of 1 represent an obstacle and 0 represents clear space. The occupancy grid is run through a particle filter to remove spurious points. An example of this is shown in figure 5.1. The occupancy grid is then passed on to the Obstacle avoidance algorithm.

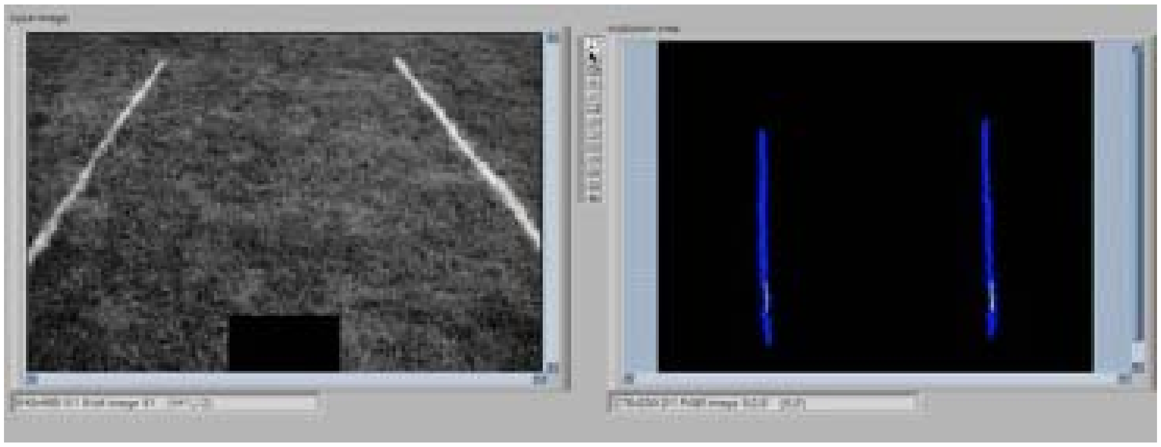

Figure 5.1- Vision Processing for IGVC Autonomous Challenge

All the data from the sensors must be put into a common format to be processed. Both the forward facing laser range finder and the occupancy grid form the camera give binary data. That is, all obstacles are impassible. All the obstacles are treated as points with a safety radius around them. The safety radius is how far away the vehicle should remain from the object. Since the obstacles are impassible, they are given a full cost of 100. The obstacles are stored in a table giving their XY location relative to the vehicle, their safety radius and their cost.

For the 3D data used by Cliff, the cost of a cell in the map is proportional to the result of the edge detection filter. The safety radius is the approximate size of the cell plus the safety range. The obstacles are again stored in a table giving their XY location relative to the vehicle, their safety radius and their cost.

Data in the tables are then filtered to remove obstacles that are too far away. Setting the threshold distance is critical to successful navigation. Setting it too far means that all paths will show up as blocked, because over a longer range, chances are an obstacle will block a path. Setting it too close will mean the vehicle cannot react to obstacles in time or will get stuck in dead ends. For the IGVC course, with a maximum speed of 5 mph, the range was set to 2 meters from the front of the vehicle. The quicker DARPA Grand Challenge race realized optimum results by setting a 7 meter range from the front of the vehicle.

5.2-Path Generation

A set of possible paths that can be taken by the vehicles must be generated. How the paths are generated depends on the geometry and steering configuration of the vehicle. The examples in this thesis consider the common cases of Ackerman steering and differential drive. For an Ackerman steered vehicle, the possible paths are the paths generated by taking the entire steering range of the vehicle and dividing it into small increments, typically no greater than 5 degrees. The steering angles are then used to calculate the turn radii for the paths. For a differentially driven vehicle, a polar array of vectors between negative 90° and positive 90° of some pre-selected length (usually around 1 vehicle length) is created in front of the vehicle. These are typically set with a spacing of 5 degrees. Each vector then defines the start point and end point of a circular arc. The tangents to these arcs must be in the current direction of vehicle motion at the beginning of the arc. The arc paths intersect with the tip of the vector. This is illustrated in Figure 5.2. The shorter the vector, the smaller the specified turning radius of the vehicle and the more aggressively the vehicle will drive. The path pointing straight ahead has a radius of infinity and is therefore treated as a special case.

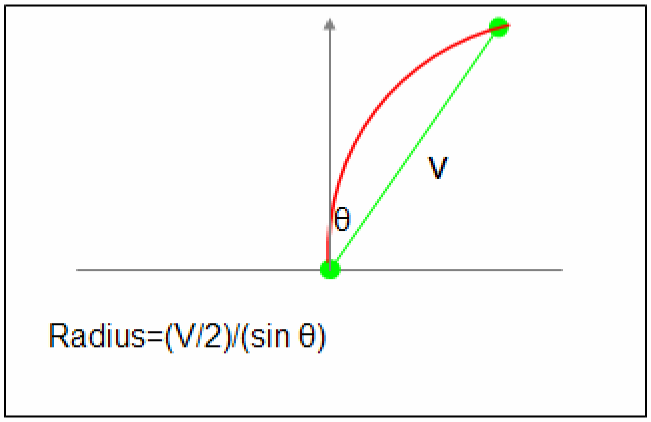

Figure 5.2- Vector to Arc Path Conversion

Each path is numbered (i) from left to right from 1 to n where n is the number of paths. Each path also has a cost associated with it. The initial cost of the path is zero. This will be modified based on the obstacles and on information from the global path planner.

5.3- Obstacle Processing

Each obstacle is checked against each path to see if the obstacle interferes with the path. Figure 5.3 illustrates how this is done. The object radius is the distance from the point of turning to the object. If this is between the inner and outer radius of the path then the obstacle interferes with the path. If an object blocks a path and its cost is higher then the current cost of the path, then the cost of the path is replaced with the higher one. Once all the obstacles are checked against the path for occlusion, the cost of the path will be that of the highest costing obstacle in the path. This is illustrated in figure 5.4. This cost is called the obstacle cost of the path.

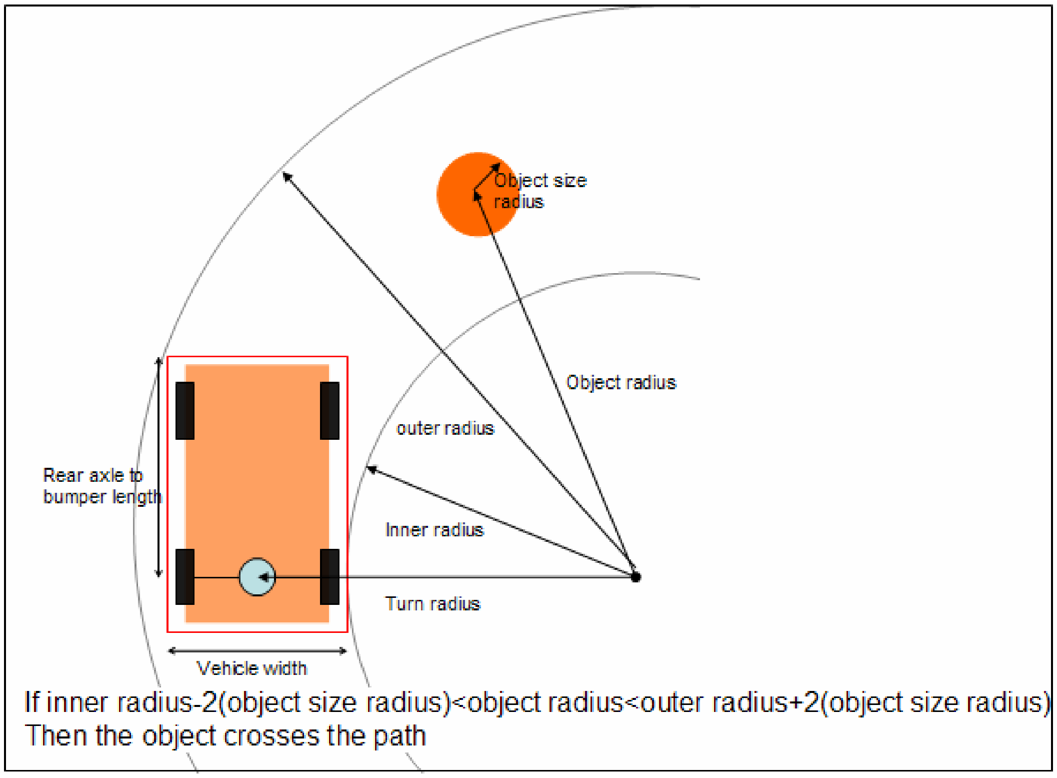
Object size radius
Object radius
outer radius
Rear axle to bumper length
Inner radius
Turn radius
Vehicle width
If inner radius-2(object size radius)<object radius<outer radius+2(object size radius)
Then the object crosses the path

Figure 5.3- Obstacle Crossing Path Check

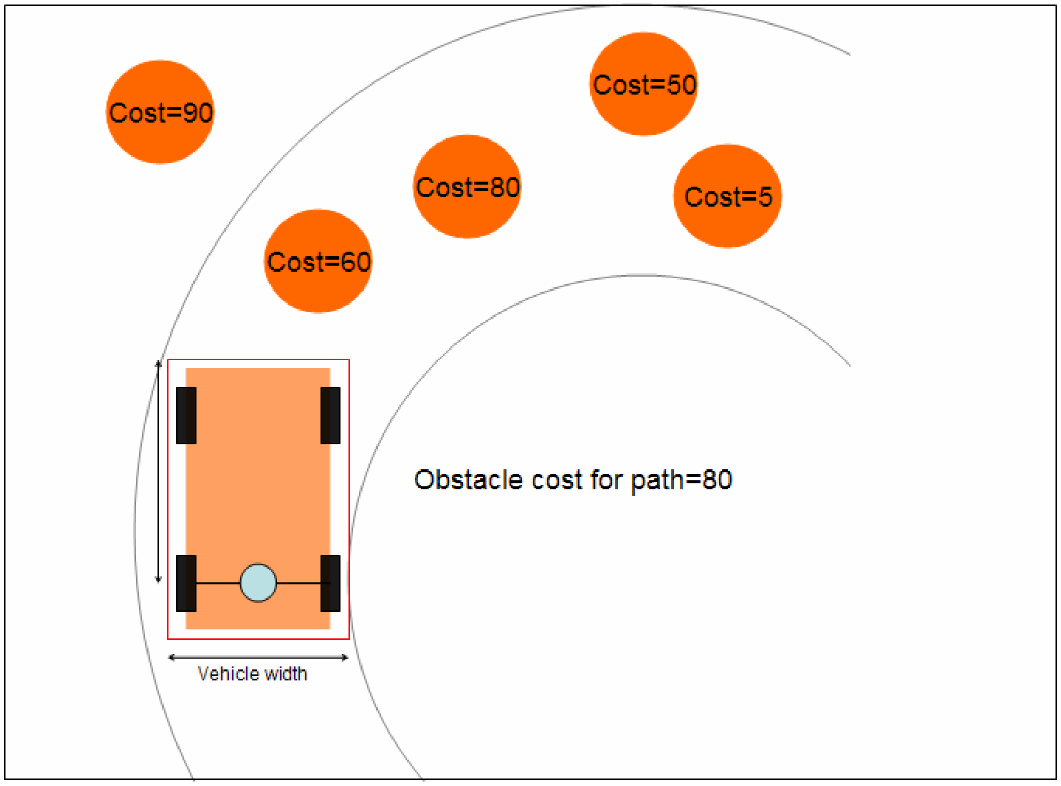

Figure 5.4- Obstacle Cost of Path

5.4- Path Planning

The path planner is what guides the robot towards the objective. For waypoint challenges such as the Grand Challenge and the IGVC Navigation challenge, the path planner steers the vehicle towards the next waypoint. For Ackerman steered vehicles the ideal path is the one with the steering angle pointing towards the next waypoint. For a differentially driven vehicle, the ideal path is the one whose θ (see figure 5.2) is the angle to the next waypoint. For the IGVC autonomous challenge the ideal path is the one whose vector V (see figure 5.2) ends in between the two white lines [7].

5.5-Final Path Selection

A path is selected based on the obstacle cost of the path, the deviation from path commanded by the global planner, and the deviation from the current path the vehicle is taking. This is done using the following objective function where 'i' represents the path number, 'ip' is the number of the path recommended by the path planner, and 'io' represents the current path number of the vehicle

$$\text{Final cost for path (i)=obstacle cost(i)} + K1*|ip-i|^2 + K2*|io-i|^2 \qquad (5.5.1)$$

The path selected is the one with the lowest Final Cost. Obstacle cost is the cost of the highest costing obstacle that crosses the path. It prioritizes paths with lower costs. |ip-i| is the deviation from the path recommended by the path planner. |io-i| is the deviation from the path the vehicle is currently taking. K1 and K2 are constants that must be tuned. Increasing K1 will force the algorithm to follow the path planner more closely even if the terrain is more difficult to cross. Increasing K2 will stop the effect of noise, or small changes in cost from causing large changes in steering angles over short distances. In general, increasing the value of K2 makes the vehicle run more smoothly. The values of K1 and K2 must be kept small enough such that they do not override the cost of a path with a non traversable obstacle.

In cases where all the paths are impassible, the IGVC vehicles were programmed to do a zero radius turn in the last good direction. This was not done on the Grand Challenge vehicle since it is incapable of zero radius turns and is incapable of reversing autonomously.

The algorithm is designed to run continuously to give a constantly updated path. The algorithm was tested at 5 Hz, 10 Hz and 15 Hz. The values of K1 and K2 did not require re-tuning for different refresh rates and there was no noticeable difference in performance at low speeds. Theoretically the program lag time will become significant at high speeds.

Chapter 6- Software Implementation

This chapter describes the how the arc path algorithm has been implemented in software. This varies depending on the number of sensors and the number of computers.

6.1-Subprograms

The software can be broken down into basic subprograms. The first subprogram, called 'path preparation,' generates a set of possible paths based on the current vehicle position and sensor readings as described in chapter 5, section 2 . Each path is given a path number, a radius, and an initial cost of 0.

The next subprogram, called 'sensor acquire and process,' reads the obstacle data from a sensor and formats them as described in chapter 5 section 1. One of these sub programs is required for every sensor

The third subprogram, named 'obstacle processing,' checks each path against each obstacle and updates the path cost as described in chapter 5 section 3.

The forth subprogram is called 'final path selection.' It selects a path based on the criteria described in chapter 5 section 5.

The next sub program is the 'path planner,' which provides the 'final path selection' with a desired path heading in order to guide to robot to its final destination. This is described in chapter 5 section 4.

6.2-Single Sensor implementation

Figure 6.1 shows how the subprograms interact when only a single sensor is used. The subprograms are divided such that they can be implemented on separate computers connected via a network.

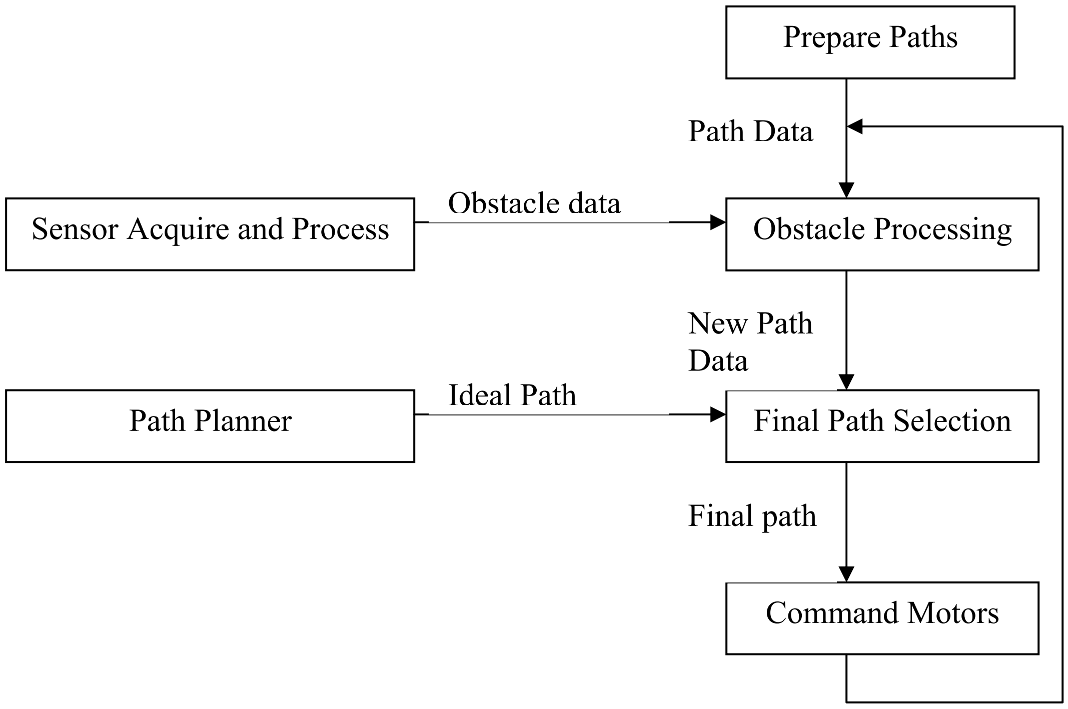

Figure 6.1- Single Sensor Implementation

6.3-Multiple sensor implementation

The algorithm is well suited for multiple sensors of any type and combination due to the standardized obstacle format. There are several ways multiple sensors can be implemented, 1 of which are shown in figure 6.2

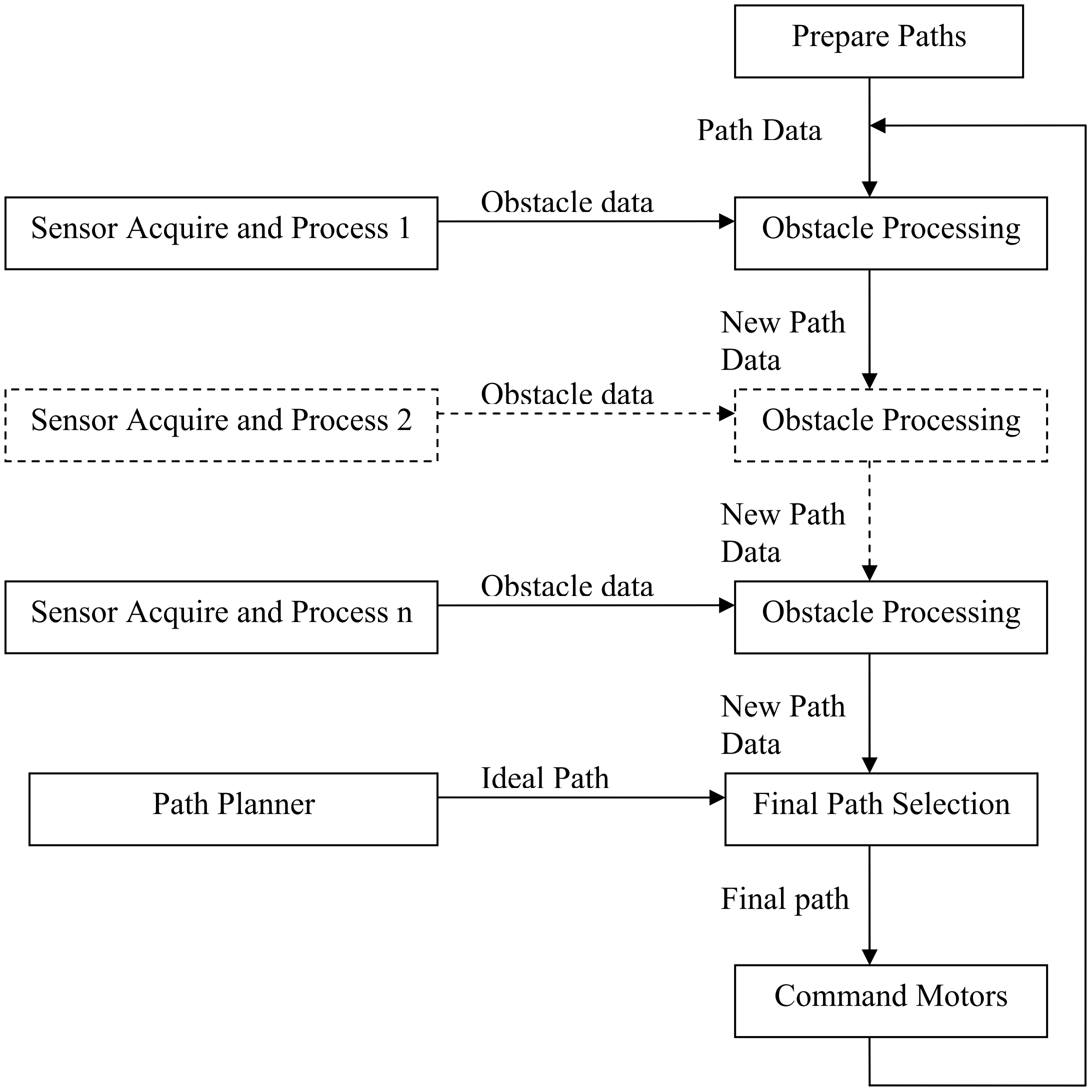

Figure 6.2- Multiple Sensor Single Thread Implementation

Chapter 7-Performance and Conclusion

7.1-Performance

At the 2004 IGVC, the obstacle avoidance algorithm performed well during the Autonomous Challenge since none of the vehicles ever hit any obstacles. The vehicles however, did get trapped in several complex obstacle configurations. The vehicles were able to extricate themselves from these traps by performing a zero radius turn in the last good direction. The differential drive vehicles were fully capable of negotiating the complex obstacles configurations, but none of the arc paths would show up as clear. This was solved by reducing the range of considered obstacles from 2 meters to 1 meter; this way all the paths would not show up as impassible. With the range reduced to 1 meter, the top speed achievable was 3mph. Beyond this the vehicles would collide with obstacles. This is due to acceleration limitations of the motors which meant the vehicles path could not be changed in time to avoid the obstacle.

At the 2004 Grand Challenge Qualifier event, Cliff finished the qualifier course at a speed of 3mph without hitting any obstacles. When the speed was increased to 5mph, the obstacle avoidance continued to work flawlessly but the vehicle failed to finish the qualifier course due to electrical problems. At 7mph the obstacle avoidance began to fail. It would side swipe obstacles and in some cases come to a complete stop in front of the obstacle. This was because the steering angle could not be changed in time to avoid the obstacle. It was later shown in testing that increasing the range of obstacles from 7 meters to 15 meters meant the obstacle avoidance would work at speeds up to 10mph but this would result in the vehicle being trapped when confronted with complex obstacle configurations or sharp turns.

7.2-Conclusions

The Arc path algorithm performs roughly the same as the curvature velocity method but with less computations and the ability to handle non-binary obstacles. The performance of the algorithm depends largely on the range at which obstacles are considered and the dynamic effects of the vehicle. Setting the range too far results in the algorithm trapping itself in scenarios which are otherwise traversable. Setting the range too close results in the vehicle not being able to react to obstacles in time. This is due to the vehicles dynamic limitations. Setting the range too low can also result in the vehicle getting trapped in true dead ends. In general the closer the obstacle range, the lower the speed of the vehicle has to be set. Because the algorithm does not consider vehicle dynamics, the top speed at which the algorithm can be used will always be limited. Additional work to account for vehicle dynamics could significantly improve the performance of the algorithm.

The algorithm is well suited for both differentially driven vehicles and Ackerman steered vehicles. The algorithm is not well suited for tracked vehicles or skid steer vehicles. This is because wheel slip causes the vehicles to deviate sharply from the arc path. Instead, tracked vehicles are best restricted to zero radius turns and forward motion which can be controlled fairly accurately. Therefore tracked and skid steer vehicles are well suited to an algorithm like the Vector Field Histogram (VFH).

The Arc path algorithm is also easily integrated with any path planner. All the path planner needs to provide is a preferred arc for the vehicle to travel along. How closely the algorithm follows the path planner is selectable.

The algorithm has proved sufficient for use on the IGVC. The algorithm also performed adequately at the 2004 DARPA Grand Challenge Qualifier. However higher speeds are required for the actual DARPA Grand Challenge course. More complex terrain configurations are also expected that will likely trap the vehicle. It is likely that a more advanced algorithm, which accounts for vehicle dynamics, will be required to achieve the speeds necessary to complete in events such as the DARPA Grand Challenge. A more advanced path planner may also prove necessary to complete the DARPA Grand Challenge

7.3-Future Work

Since the range at which obstacles are considered plays such an important role in the performance of the algorithm, it would be helpful to implement dynamic range selection. This would allow the range to be shortened in regions with densely packed obstacles requiring low speeds, and it would allow the range to be extended when no obstacles are present and higher speeds can be achieved.

I would also like to create a more advanced Path Planner for the waypoint challenges. Steering directly towards the next waypoint is not always the best way to get there. I would like to use a shortest path algorithm such as the A*/D* algorithms to guide the robot over the entire range of the vehicle sensors. We currently rely completely on the collision avoidance algorithm to steer the vehicle clear of dead ends and to negotiate its way around complex obstacle scenarios. In essence, the collision avoidance algorithm itself is still being used as a path planner as opposed to a purely collision avoidance algorithm. With a more advanced path planner that will plan paths around dead ends and through complex obstacle configurations, the collision avoidance algorithm can be tuned purely for collision avoidance. The complexity of the path generated by the A*/D* algorithms can be used as a measure of obstacle complexity, which in turn could be used to suggest a safe vehicle speed. This safe vehicle speed can be used to influence the dynamic range selection mentioned earlier.

As mentioned earlier, the Arc path algorithm would be improved if it could be modified to account for vehicle dynamics. One possibly effective modification would be a look-ahead function that would modify the algorithm based on where the vehicle will be after a certain time step. This could be based on the vehicles previous trajectory. Such an approach would help account for the time lag from sensor acquisition to motion command and also the reaction time of the vehicle. Another modification would be to incorporate a Dynamic Window Approach, but this would also require a more advanced path planner. In other words, the algorithm would only select paths achievable in the next frame. The common disadvantages of a dynamic window approach such as getting stuck in dead ends or local minima would be mitigated, because the advanced path planner would guide the vehicle away from dead ends.

REFERENCES:

[1] J. Borenstein, Member, IEEE and Y. Koren, *"The Vector Field Histogram"*, IEEE Journal of Robotics and Automation Vol 7, No 3, June 1991, pp. 278-288.

[2] Y. Koren, and J. Borenstein, *"Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation*", Proceedings of the IEEE Conference on Robotics and Automation, Sacramento, California, April 7-12, 1991, pp. 1398-1404

[3] Iwan Ulrich and Johann Borenstein, "*VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots*", Proceedings of the 1998 IEEE International Conference on Robotics and Automation. Leuven, Belgium, May 16–21, 1998, pp. 1572 - 1577

[4] Alonzo Kelly and Anthony Stentz, *"Analysis of Requirements for High Speed Rough Terrain Autonomous Mobility Part II: Resolution and Accuracy",* Submitted to the 1997 IEEE International Conference on Robotics and Automation. Albuquerque, New Mexico, April 1997

[5] D. Coombs, K. Murphy, A. Lacaze, S. Legowik, *"Driving Autonomously Offroad up to 35 km/h",* Proceedings of the 2000 Intelligent Vehicles Conference, the Ritz-Carlton Hotel, Dearborn, MI,USA, October 4-5, 2000.

[6] Reid Simmons, *"The curvature velocity method for local obstacle avoidance",* Proceedings of the International Conference on Robotics and Automation (ICRA), p3375-3382, 1996

[7] IGVC Rules, available: http://www.igvc.org/deploy/rules.htm

[8] Robin R. Murphy, "*Introduction to AI Robotics*", The MIT Press, 2000

[9] D.H. Shin and S. Singh, "*Path Generation for Robot Vehicles Using Composite Clothoid Segments*", tech. report CMU-RI-TR-90-31, Robotics Institute, Carnegie Mellon University, December, 1990.

[10] M. Pivtoraiko, "*A Study of Polynomial Curvature Clothoid Paths for Motion Planning for Car-like Robots*", tech. report CMU-RI-TR-04-68, Robotics Institute, Carnegie Mellon University, December, 2004.

Appendix A- Matlab Script to Generate a Clothoid, Programmed by Andrew Bacha

```
%clothoid test
clear all;

%vehicle parameters
r=1.26;      %rear track, m
w=2.11;      %wheel base, m

%vehicle dynamics
maxturnvel=31;  %deg/s
steermax=40;    %deg  ALWAYS POSITIVE!!!!!!!!!!!
under=1;        %NIST understeer coeff

%to apply properly, solve at each dt
%init cond
steer(1)=15;      %deg, positive is right

%command
speed=20;       %mph
steerdot=-31;      %deg/s

%units
steer(1)=steer(1)*pi/180;   %deg -> rad
steerdot=steerdot*pi/180;   %deg/s -> rad/s
%convert from grand challenge convention (pos degrees is to right)
%to my calculations (pos degrees is to left)
steer(1)=-1*steer(1);
steerdot=steerdot*-1;
steermax=steermax*pi/180;   %deg -> rad
speed=speed * 0.44704;  %mph -> m/s

%trajectory parameters
smax=10;   %curve length

%init conds
x(1)=0;
y(1)=0;
h(1)=pi/2;    %initial heading
s=0;
dt=0.005;      %time step

count=1;

%Solve states at each step
while (s<smax)
    %solve for velocities
    xdot=cos(h(count))*cos(steer(count))*speed;
    ydot=sin(h(count))*cos(steer(count))*speed;
    hdot=(under/w)*sin(steer(count))*speed;
    %apply velocities
    count=count+1;
    x(count)=x(count-1)+xdot*dt;
    y(count)=y(count-1)+ydot*dt;
    steer(count)=steer(count-1)+steerdot*dt;
    if (abs(steer(count)) > steermax)
        steer(count)=steermax*sign(steer(count));
    end
    h(count)=h(count-1)+hdot*dt;
    s=s+speed*dt;
end
```

```
%adjust position so front wheels
%are on 0,0
y=y-w*sin(h(1));
x=x-w*cos(h(1));

%find position of left/right side of vehicle
xr=x+(r/2)*cos(h-pi/2);
xl=x-(r/2)*cos(h-pi/2);
yr=y+(r/2)*sin(h-pi/2);
yl=y-(r/2)*sin(h-pi/2);

%find positions of front of vehicle
xfr=x+w*cos(h)+(r/2)*cos(h-pi/2);
xfl=x+w*cos(h)-(r/2)*cos(h-pi/2);
yfr=y+w*sin(h)+(r/2)*sin(h-pi/2);
yfl=y+w*sin(h)-(r/2)*sin(h-pi/2);

figure(1);
plot(xr,yr,'b');
%axis([-1 3 0 4]);
hold on;
plot(xl,yl,'b');
plot(xfr,yfr,'r');
plot(xfl,yfl,'r');
hold off;

figure(5);
plot(xr,yr,'b');
hold on;
plot(xl,yl,'b');
plot(xfr,yfr,'r');
plot(xfl,yfl,'r');
axis([-8 8 -2 14]);
steerdot

%trim back and front to match
%it also helps keep end points nice and paired
xr=xr(1,round(w/(dt*speed))+1:size(xr,2));
xl=xl(1,round(w/(dt*speed))+1:size(xl,2));
yr=yr(1,round(w/(dt*speed))+1:size(yr,2));
yl=yl(1,round(w/(dt*speed))+1:size(yl,2));

xfr=xfr(1,1:size(xfr,2)-round(w/(dt*speed)));
xfl=xfl(1,1:size(xfl,2)-round(w/(dt*speed)));
yfr=yfr(1,1:size(yfr,2)-round(w/(dt*speed)));
yfl=yfl(1,1:size(yfl,2)-round(w/(dt*speed)));

figure(2);
plot(xr,yr,'b');
%axis([-1 3 0 4]);
hold on;
plot(xl,yl,'b');
plot(xfr,yfr,'r');
plot(xfl,yfl,'r');
hold off;

%save cloth.mat xr yr xl yl xfr yfr xfl yfl -mat
```