Advanced Path Planning for an Autonomous SAE Electric Race Car

Martin French 20743871

2014 FYP

Project Report

Supervisor: *Professor Dr. Thomas Bräunl*
Submitted: Friday 24th October 2014

The REV Project
The University of Western Australia

The University of Western Australia
Achieving International Excellence

Abstract

This project focuses on the implementation of local path planning for the UWA REV Autonomous SAE Electric Race Car, implementing a solution amongst existing code in a practical way. The implemented advanced path-planning algorithm is built and based on an algorithm used by both Hanyang University in Korea and Harvard University in the US, competing in separate autonomous vehicle challenges. In essence, the algorithm will generate a range of possible manoeuvres, assess each option using a combination of cost functions and finally select the manoeuvre with the least cost to submit to vehicle control.

The structures of the advanced path planning algorithm and the simulator used for testing are discussed, with results presenting the correct functionality and also the limitations of the implementation when tested in the simulator and on the UWA REV Autonomous SAE Electric Race Car. The findings conclude that the implementation successfully provides navigation around static obstacles, thus achieving the goal of this project.

Acknowledgements

The author would like to thank the following people for their guidance, support and contributions throughout the duration of the project.

Thomas Bräunl – Project Supervisor,
Thomas Drage – AutoSAE Team Leader,
Ruvan Muthu-Krishna – AutoSAE Team Member,
Family and Friends.

Advanced Path Planning for an Autonomous SAE Electric Race Car

PROJECT REPORT

Table of Contents

1.0 Introduction

Autonomous cars are vehicles capable of navigating through an environment, often self-sensed, without human intervention. These types of cars commonly exist as prototypes and platforms for learning in the area of robotics. From the 1920s through to the 1950s, researchers produced primitive ‘driverless’ prototypes that would lay the foundations for autonomous car technology for the future [1]. These cars were radio controlled or could follow wires/electrical circuits embedded in the roadways [2]. It wasn’t until the 1980s before large companies like Mercedes-Benz and universities such as the Bundeswehr University of Munich developed something that resembles modern autonomous technology. Their Eureka Prometheus Project was a vision-guided vehicle, which attained a top speed of 39mph on the streets without traffic [1]. As computing power increased the complexity of autonomous driving systems followed with more public projects such as the Park Shuttle public transport system in the Netherlands in the early 2000s [3].

The only commercially fully autonomous vehicle available today is an electric shuttle bus which travels at speeds just over 12mph, ideal for contained environments such as universities and corporate campuses [4]. Although fully autonomous cars are not yet readily available to the public, several manufacturers produce cars with driver assist systems for increased safety, fuel economy and driver comfort. Systems of this type include Audi’s adaptive cruise control [5], which automatically adjusts speed to keep a safe distance from the car in front and Mercedes lane keeping system, which steers the vehicle within it’s lane [6].

Recently, autonomous performance cars have been of interest to researchers as a means to push the boundaries of automotive racing that is limited by the abilities of human drivers. One such example is Stanford University’s Audi TT ‘Shelley’ which has proven itself by posting lap times comparable to that of a seasoned driver [7]. Stanford has also developed an off-road autonomous vehicle ‘Stanley’ for the DARPA Grand Challenge [8]. Similarly, Hanyang University in Korea developed an autonomous off-road Hyundai ‘A1’ to compete in the Autonomous Vehicle Competition (AVC) organized by the Hyundai–Kia Automotive Group [9]. These three vehicles are of particular interest to this project because they operate in a non-urban scenario without traffic. This type of autonomous challenge best fits the goals and operating scenario of the UWA Autonomous SAE Electric Race Car, which will be subsequently referred to as the AutoSAE.

Path-planning solutions have been heavily researched in the past with a large emphasis on robotic applications. Consequently, many of the proposed algorithms are undesirable for automotive applications. Path planning and vehicle navigation can be broken up into two steps; First: *Global path planning*, which deals with generating the most ideal route to reach the final destination and second: *Local path planning*, which is responsible for obstruction avoidance and path planning for each segment generated by the global path planner [10]. This project is more concerned with local path planning and obstruction avoidance as a global set of waypoints are provided to the system pre-operation. Proposed methods of local path planning primarily fall into three sub-divisions: potential field approaches, grid based approaches, and discrete optimization approaches.

The potential field approach to path planning involves the generation of an artificial vector field influenced by obstacles and waypoints [11]. Obstacles to be avoided are represented by repulsive potentials while the goal or waypoints are represented by an attractive potential. This approach is computationally less expensive compared to commonly proposed search algorithms however doesn't always yield a path to the goal because of local minima in the vector field. Stanford's Audi TT 'Shelley' uses this method for lane keeping and obstacle avoidance at the limits of vehicle capability [7].

Grid based approaches such as the A* search algorithm outlined by Ozguner et al. in [10] are based on a distance-cost heuristic function to determine the order of searched nodes on a graph. Consider a map diced into equally sized tiles, each tile can be seen as a node with a line connecting adjacent tiles as an edge on the graph. A cost is associated with traversing an edge and the algorithm tries to minimize the total cost of traveling from the initial node to the goal node. While this method is ideal for robots with high manoeuvrability operating in cluttered environments, it does not translate well to lane/track based driving with few obstacles.

A third path planning method is the discrete optimization approach used by both Stanford's 'Stanley' [8] and Hanyang's 'A1' [9]. This approach computes a finite number of paths generated from a set of equations describing vehicle motion. The path is defined by the lateral offset perpendicular to a fixed base frame. A cost is calculated for each path based on various criteria such as path 'smoothness', path consistency and path safety. Finally, the path associated with the least cost is selected. The discrete optimization method is best suited for this project because of its ability to generate paths parallel to a base frame (the track trajectory), resulting in real time performance gains. This method also proves to be favorable because it's generated paths match the geometry of movement for a front wheel steer automobile such as the AutoSAE used in this project. As such the implemented algorithm will be guided by the work of Hanyang University and their algorithm developed for 'A1'.

This project focuses on the implementation of local path planning for the UWA REV Autonomous SAE Electric Race Car, amongst existing code in a practical way with tests, rather than algorithm improvements. Accordingly, the goal of this project is to implement a path planning solution capable of navigating static obstacles to be used within the existing car code. The remainder of this report will be broken down as follows: The existing systems on the AutoSAE will be briefly introduced in Section 2.0. An outline of the algorithm, including base frame generation, path candidate generation, path cost calculation and path selection will be presented in Section 3.0. The testing system, results and recommendations will be discussed in Section 4.0 and the report concluded in Section 5.0.

2.0 System Overview

The AutoSAE uses three main control components: the Control Software, the Low Level Controller and the Safety Supervisor. The Control Software concerns itself with environment mapping, path planning, vehicle control as well as communicating with an emergency base station and a webpage for human-machine interfacing. The Low Level Controller takes commands from the Control Software to actuate physical components on the car including the steering wheel and the hydraulic brakes. The Low Level Controller also provides a signal to the motor controllers for vehicle acceleration. The Safety Supervisor provides an additional layer of safety should the Control Software fail. These three systems were installed on the car as part of previous projects [12][13]. Figure 1 below illustrates the relationship between the three systems.

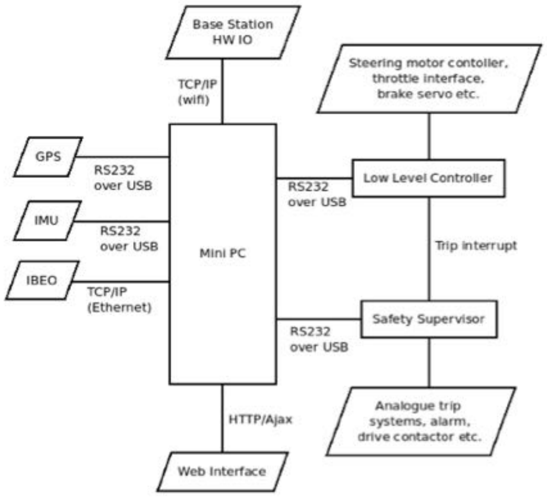


FIGURE 1 – 'AUTOSAE' SYSTEM OVERVIEW. SOURCE [12]

The Control Software collects information from an array of sensors and interfaces including a GPS, Inertial Momentary Unit, an IBEO Laser Scanner and a Web Interface, subsequently building a map of waypoints, obstacles and current position [12]. The algorithm implemented in this project is incorporated between this mapping task and vehicle control task as illustrated in Figure 2 below.

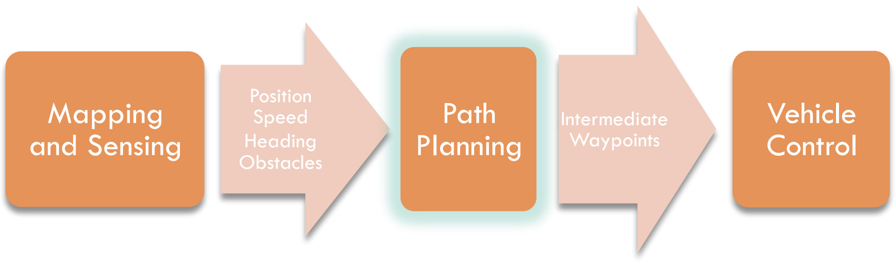


FIGURE 2 - CONTROL SOFTWAE FLOW

The task of the Vehicle Control component is to take a set of immediate waypoints from the Path Planning algorithm, calculate the desired steering angle and speed, subsequently commanding the Low Level Controller. Vehicle control is a crucial module for testing the implementation of this project so it is beneficial to understand how it operates. When passed a set of waypoints, the vehicle control will sequentially drive to each point until it reaches the final one. A waypoint is marked as reached when the vehicle has come within a defined tolerance from its position. A heading to the next waypoint is calculated and set as the desired bearing. The steering command to reach this desired bearing is calculated using a PID loop with the desired bearing as its set point and the current heading as the process variable. The PID loop is tuned such that the heading of the vehicle will align itself with the desired bearing, consequently steering the vehicle toward the next waypoint.

The path-planning algorithm implemented in this project is initially developed in MATLAB for rapid development and simplicity of debugging. The various algorithms and functions are first developed and tested in MATLAB before being converted to C++ code using the MATLAB coder. This code is subsequently compiled and archived into a static library for use in the existing car control code.

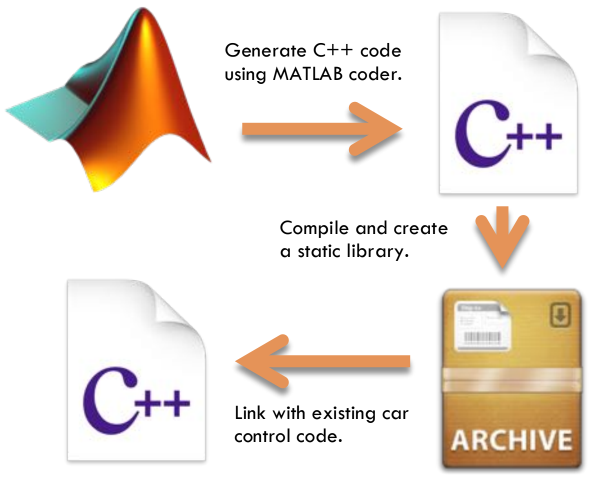

Generate C++ code using MATLAB coder.
C++
Compile and create a static library.
ARCHIVE
Link with existing car control code.
C++

FIGURE 3 - CODING APPROACH

The multi-threaded nature of the existing control code made the implementation simple. A new thread is generated once the first waypoint is hit, if advanced path planning is desired. This code conceptually runs along side the GPS/IMU processing thread using a mutex to ensure the path plan doesn't change while calculating a new heading or sending commands to the low level system.

A list of MATLAB functions and their descriptions can be found in Appendix A1.

3.0 Algorithm Description

This section of the report will outline the mathematical concepts of the path-planning algorithm, which is broken up into the following four sub tasks: base frame generation, candidate manoeuvre generation, manoeuvre cost calculation and manoeuvre selection.

Before run time, the base frame is generated using a parametric cubic spline through a set of given waypoints. Then, at each path-planning interval, the position of the vehicle is determined on the base frame enabling the extraction of the directional information embedded in the base frame. Next, a set of path candidates is generated from the integration of a vehicle model using the base frame and current vehicle heading. From each path in this set, a cost is calculated with respect to three cost criteria: base frame offset, path consistency and path safety. Finally the path associated with the least cost is selected and submitted to Vehicle Control.

3.1 Base Frame Generation

As mentioned above the base frame is a parametric cubic spline of waypoints that provides directional information for the path-planning algorithm. A parametric cubic spline parameterized by an arbitrary parameter is not ideal for path generation and motion control. Consequently, an arc length parameterized cubic spline is generated to simplify path generation and enable the algorithm to perform in real time [9].

To achieve this, a parametric cubic spline is generated from the given waypoints using the minimal centripetal parameterization proposed by Lee [14]. The total path length is then calculated using the adaptive Gaussian quadrature numerical integration method developed by Guenter and Parent [15]. The curve is then broken up into equidistant points with the separation of each point defined as a design parameter stipulating how well the arc-length parameterized curve should match the original curve. The 2-D arc-length parametric cubic spline is defined in Equation 1; with *s* representing the arc length along the base frame and the subscript *i* denoting the spline segment number [16].

$$\begin{aligned} x_b(s) &= a_{x,i}(s-s_i)^3 + b_{x,i}(s-s_i)^2 + c_{x,i}(s-s_i) + d_{x,i} \\ y_b(s) &= a_{y,i}(s-s_i)^3 + b_{y,i}(s-s_i)^2 + c_{y,i}(s-s_i) + d_{y,i} \end{aligned} \qquad (1)$$

Subsequently, the tangent and curvature of the base frame can simply be calculated using the following equations [9]:

$$\frac{dx_b}{ds} = x_b' = \cos\theta_b \quad \frac{dy_b}{ds} = y_b' = \sin\theta_b \qquad (2)$$

$$\kappa_b = \frac{x_b' y_b'' - x_b'' y_b'}{(x_b' y_b')^{\frac{3}{2}}} \qquad (3)$$

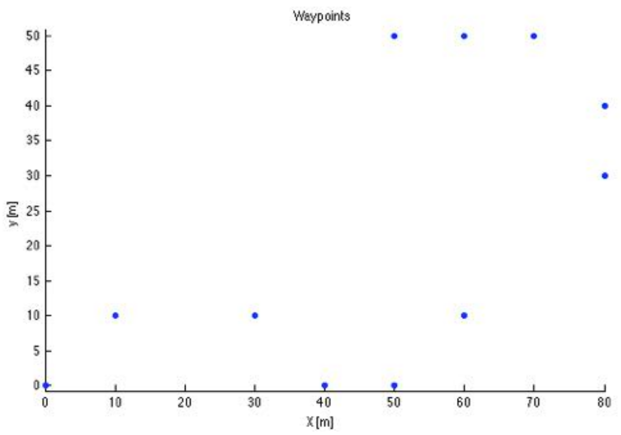


FIGURE 4 – UNEQUALLY SPACED WAYPOINTS

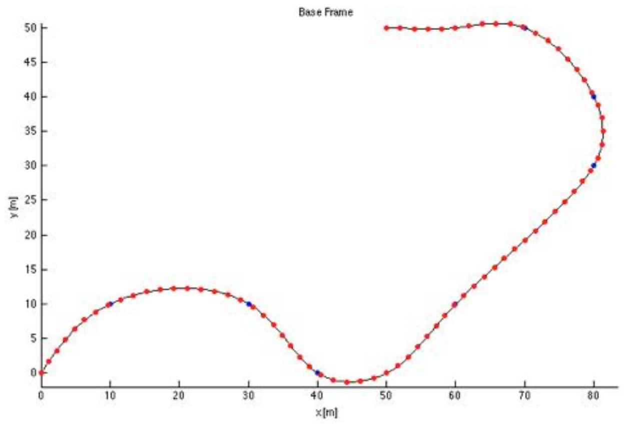


FIGURE 5 – ARC-LENGTH PARAMETERIZED SPLINE (BASE FRAME)

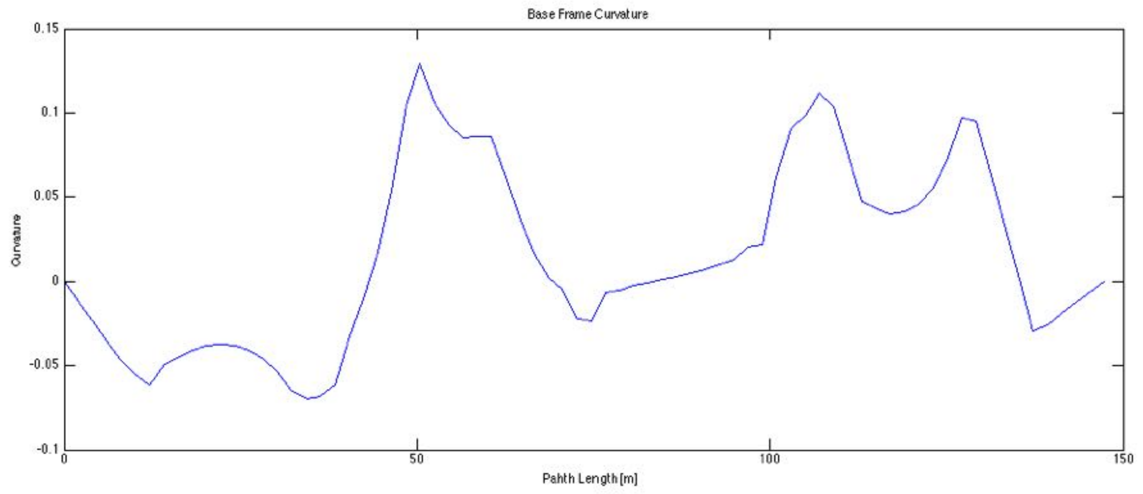


FIGURE 6 - BASE FRAME CURVATURE

$$\kappa_b = \frac{x'_b\, y''_b - x''_b\, y'_b}{(x'_b + y'_b)^{3/2}} \qquad (4)$$

FIGURE 4 above shows 11 unequally spaced waypoints that are used to develop the arc-length parameterized cubic spline shown in Figure 5. Figure 6 is a graph of path curvature against path length for the generated base frame in Figure 5, which is computed using Equation 4 [9]. Candidate paths will be generated from this curvature to utilize the directional information the base frame provides.

3.2 Localization

Clearly, the main goal of local path planning is to navigate around obstacles and choose the most ideal path at that moment. Consequently, the car will deviate from the base frame during operation. The algorithm therefore needs to localize the position of the car on the base frame so the directional information can be extracted. This is achieved by finding the closest point on the curve using a combination of Quadratic Minimization and Newton's method proposed by Wang et al [17].

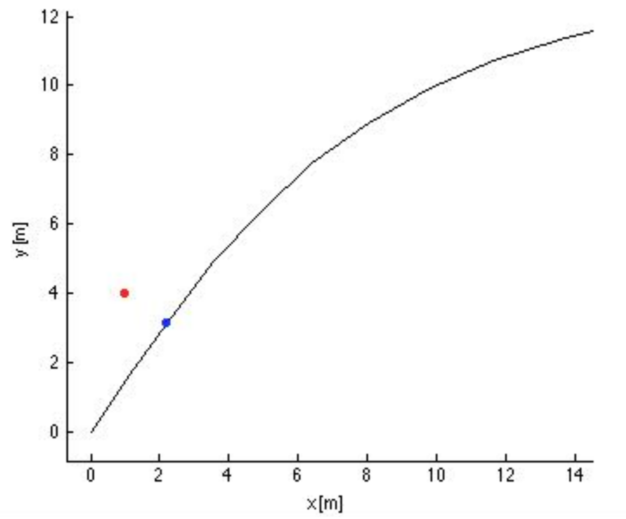


FIGURE 7 – LOCALIZATION. RED: CAR POSITION, BLUE: CLOSEST POINT ON THE BASE FRAME

3.3 Candidate Manoeuver Generation

3.3.1 Coordinate System

A curvilinear coordinate system is set up to enable simple smooth path generation [9]. Figure 8 illustrates the concept of the coordinate system and it's geometric relationship to the generated path. The distance along the base frame *s* becomes the horizontal axis and the tangential distance offset from the base frame *q* becomes the vertical axis. Figure 8 also demonstrations a generated path where the difference between the final offset q_f and the initial offset q_i is non-zero and consequently the generated path is bent further away from the base frame.

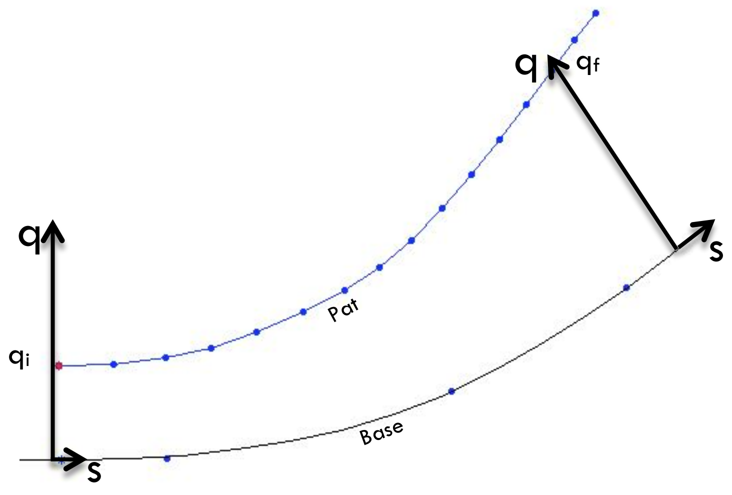


FIGURE 8 – CURVILINEAR COORDINATE SYSTEM

3.3.2 Manoeuvre Generation

Ignoring the curvature of the base frame and working within the curvilinear coordinate system, a path is defined by the length Δs_f along the base frame and its final offset q_f from the base frame. A smooth path needs to be generated to take the vehicle from its current offset *qi* to its final offset q_f. It is essential that the generated paths take into account the current heading of the vehicle (illustrated in Figure 9)

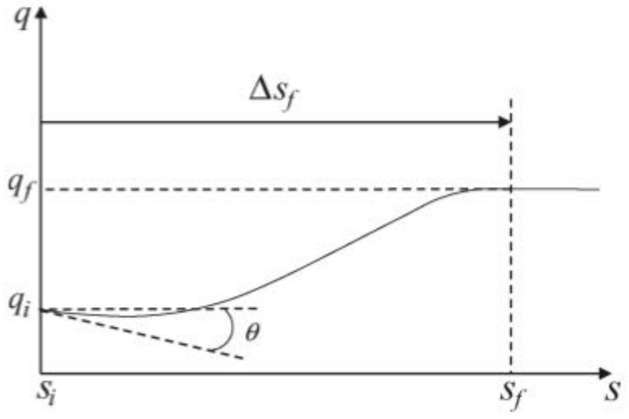


FIGURE 9 – CURVILINEAR COORDINATE SYSTEM MANOEUVRE. THETA: ANGLE DEVIATION FROM BASE FRAME. SOURCE [9]

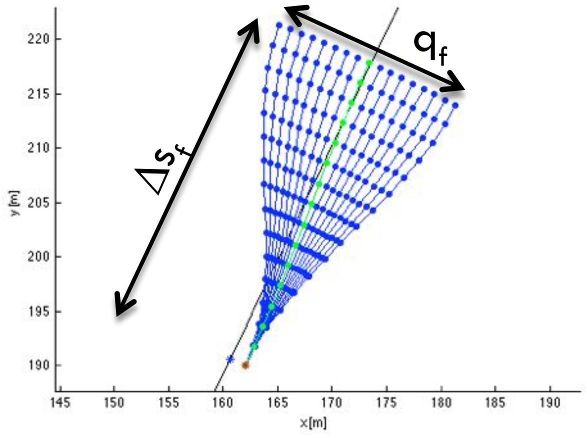


FIGURE 10 – 17 GENERATED PATHS

to ensure realistic path planning. θ represents the angle difference between the current vehicle heading and the heading of the localized point on the base frame. It is also desirable to have the end of the path align with the heading of the base frame at that point as shown in Figure 9 and Figure 10. These four requirements translate to 4 boundary conditions defined by Equations 6 below. Each manoeuvre is defined by a cubic polynomial (Equation 5), the coefficients of which can be computed using the aforementioned boundary conditions [9][18].

$$q(s) = \begin{cases} a\Delta s^3 + b\Delta s^2 + c\Delta s + q_i, & s_i \le s < s_f \\ q_f, & s_f \le s \end{cases} \quad (5)$$

$$q(s_i) = q_i \quad \frac{dq}{ds}(s_i) = \tan\theta \quad q(s_f) = q_f \quad \frac{dq}{ds}(s_f) = 0 \quad (6)$$

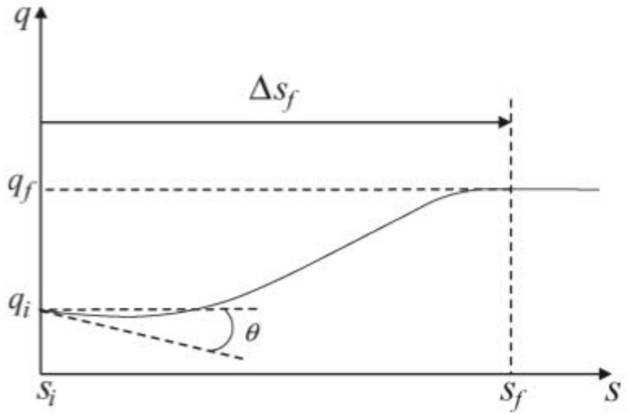


FIGURE 9 – CURVILINEAR COORDINATE SYSTEM MANOEUVRE. THETA: ANGLE DEVIATION FROM BASE FRAME. SOURCE [9]

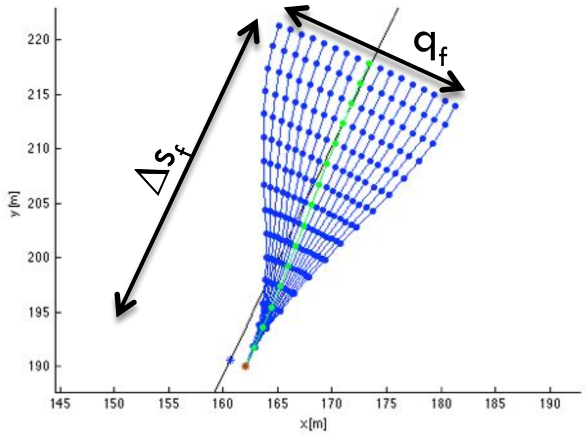


FIGURE 10 – 17 GENERATED PATHS

A set of manoeuvres, like that shown in Figure 10, are defined by their final offset q_f which are generated from a maximum lateral offset defined as a design parameter depending on the track. To further illustrate this, the paths generated in Figure 10 have a maximum lateral offset of 8m with a granularity of 1m. So the path final offsets (q_f) are therefore -8, -7, …, 0, …, 7, 8m from the base frame respectively, which covers a 16m wide track.

Once the coefficients for each manoeuvre are solved, the actual curvature of the manoeuvre combined with the base frame curvature can be computed as follows [19]:

$$\kappa = \frac{S}{Q}\left(\kappa_b + \frac{(1-q\kappa_b)\left(\frac{d^2q}{ds^2}\right)+\kappa_b\left(\frac{dq}{ds}\right)^2}{Q^2}\right) \quad (7)$$

$$S = sgn(1-q\kappa_b) \qquad Q = \sqrt{\left(\frac{dq}{ds}\right)^2 + (1-q\kappa_b)^2} \quad (8)$$

If the radius of curvature $1/\kappa$ at any point of a generated manoeuvre is less than 2m, the calculated turning radius of the 'AutoSAE', the path cannot be driven and is consequently discarded. Finally, the manoeuvres that remain are converted back to Cartesian coordinates using the following equations, first integrating for theta and subsequently x and y [9].

$$\frac{d\theta}{ds} = Q\kappa \quad \frac{dy}{ds} = Q\sin\theta \quad \frac{dx}{ds} = Q\cos\theta \quad (9)$$

3.4 Path Costs

3.4.1 Path Offset Cost

The motive for the path-offset cost is to allow the AutoSAE to run on tracks without road edges, which is a common scenario for testing. The cost aims to keep the car as close to the base frame as possible and is simply calculated as the absolute value of each paths final offset q_f. The path offset is a necessary building block for the algorithm and hence will always exist for every path-planning instance, in contrast to the safety cost, which is entirely zero when no collisions are present. This means that the path offset cost will be the final influence on deciding the ideal path if all other cost functions equal zero, and consequently ensuring an ideal path can be calculated. The path-offset cost can be expressed as:

$$C_O[i] = |q_f[i]| \quad (10)$$

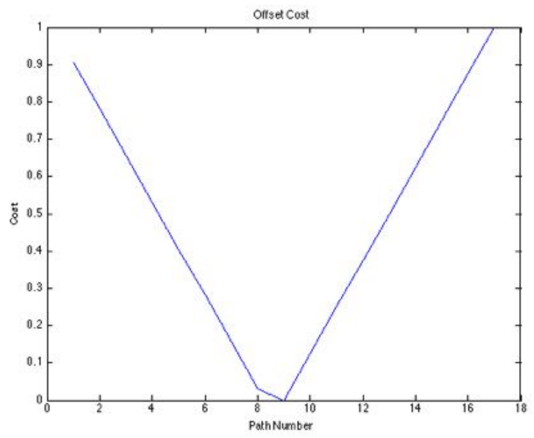


FIGURE 11 – PATH OFFSET COST

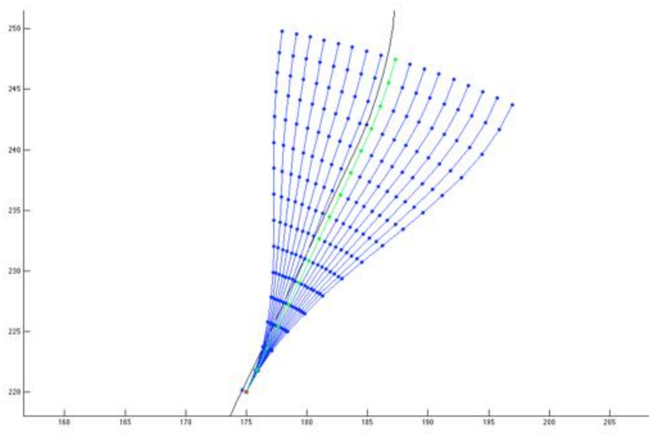


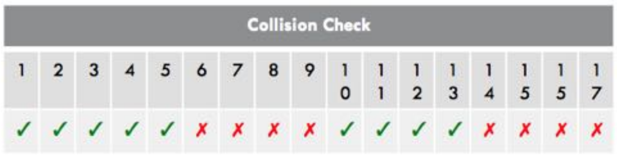
FIGURE 12 – GENERATED PATHS

Figure 11 shows a graph of offset cost for the paths shown in Figure 12. Clearly path 9 is the ideal path in this case as it is the closest to the base frame.

3.4.2 Path Safety Cost

The path safety cost is trivially part of the algorithm to ensure the chosen path does not lead to a collision. To do this, each path is checked against the map of obstacles. Each path is checked to see if it passes through a zone defined by a radius around an objects location. If it does, the path is not a feasible path and is therefore marked as having a collision. It is beneficial to invoke risk into neighboring paths to further evaluate the safety of paths that may not collide with an obstacle, but run close by. This is achieved by using Discrete Gaussian Convolution over a collision matrix [9], which produces a cost function like that shown in Figure 13. The path safety cost can be expressed as:

$$C_S[i] = \sum_{k=0}^{N} c[k]g[i-k] \qquad g[i] = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(\frac{-(\Delta q \cdot i)^2}{2\sigma^2}\right) \tag{11}$$



Collision Check																
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	15	17
✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗

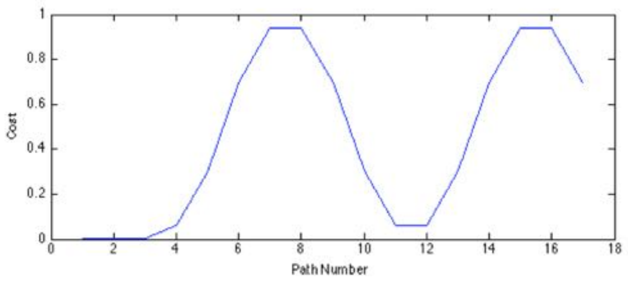


FIGURE 13 - COLLISION MATRIX AND CALUCLATED COST FUNCTION

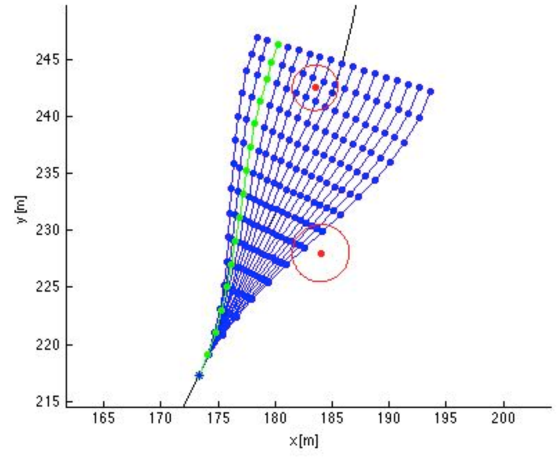


FIGURE 14 – GENERATED PATHS WITH TWO OBSTACLES

Figure 13 illustrates both the collision matrix and the associated cost function for the set of paths and their corresponding collisions with the two obstacles shown in Figure 14. Figure 13 also shows the result of the Discrete Gaussian Convolution, that there is risk in paths 5 and 10-13 even though there is no direct collision with an obstacle.

3.4.3 Path Consistency Cost

The path consistency cost provides a mechanism to choose the next path with some consideration to the previously chosen path. This cost favors paths that are most similar to the previously chosen path. The motivation for this cost is to provide a way to discern between two similarly costing paths on either side of an obstacle, like that shown in Figure 15. This situation poses a risk because choosing a path on the other side of an obstacle may result in a collision if the vehicle's control system is inadequate or the road conditions are poor.

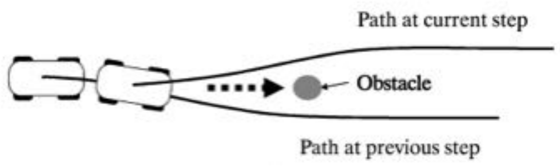


FIGURE 15 - POSSIBLE COLLISION DUE TO PATH CHANGE. SOURCE [9]

Calculating the cost associated with a given path with respect to the previously chosen path can be achieved by totaling the distance between the common arc length sections of the previous path and the new path. The arc-length common between the two paths provides suitable scaling to the cost. The equation for this cost is similar to that given in [9] with the exception that the authors use an integral where this method simply uses the sum of discrete path distances q - q_{prev}. Again, i is the path index, N corresponds to the number of common arc length points between the two paths, s_2 is the arc-length of the final common point and s_1 the initial common point, as illustrated in Figure 17.

$$C_C[i] = \frac{1}{s_2 - s_1} \sum_{k=1}^{N} \left| q_{k,i} - q_{k,prev} \right| \tag{12}$$

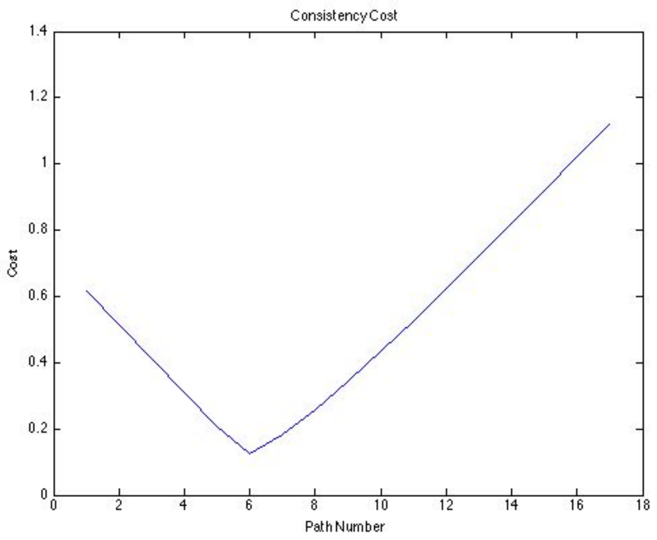


FIGURE 16 – COSISTENCY COST

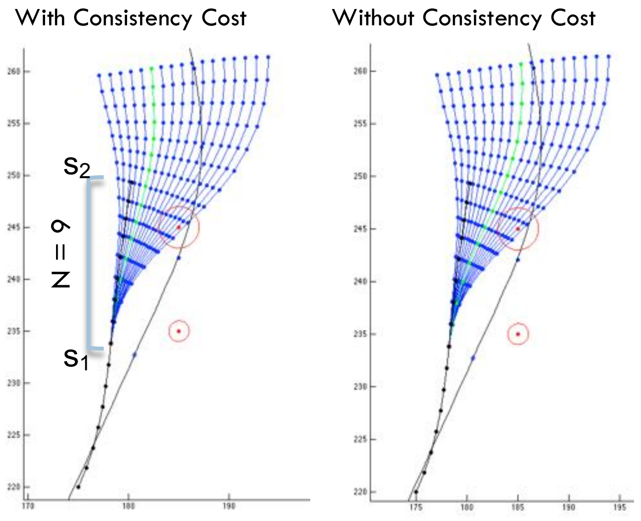


FIGURE 17 – CONSISTENCY COST PATH SELECTION COMPARISON.

Figure 16 shows the calculated consistency cost for the scenario shown in Figure 17. The effect of path consistency is evident in Figure 17 where the selected path is no longer the one that finishes closer to the base frame but aligns closer to the previous path.

3.5 Path Selection

The ideal path is calculated using a linear combination of weighted cost functions as shown in Equation 13 below. Each of the three cost functions for path offset, path safety and path consistency are scaled with a weighting w and summed for each path index i.

$$C_{TOTAL}[i] = w_O C_O[i] + w_S C_S[i] + w_C C_C[i] \qquad (13)$$

These weightings are important as they allow the operator to change the way the car drives depending on the requirements of certain scenarios. For example, using a relatively high safety weighting will choose a path further from obstacles but with a greater offset from the base frame.

The path associated with the lowest total cost is chosen as the ideal path. The Cartesian coordinates of that path are finally submitted to the vehicle control code discussed in Section 2.0 as waypoints.

3.5.1 Longest Collision Free Path

In the exceptional case where there is no collision free path available, like that shown in Figure 18, the path with the longest collision free length is chosen and submitted to vehicle control under the assumption that a collision free path will subsequently be discovered, like that shown in Figure 19. This action is sufficient enough for all scenarios intended for this implementation [9].

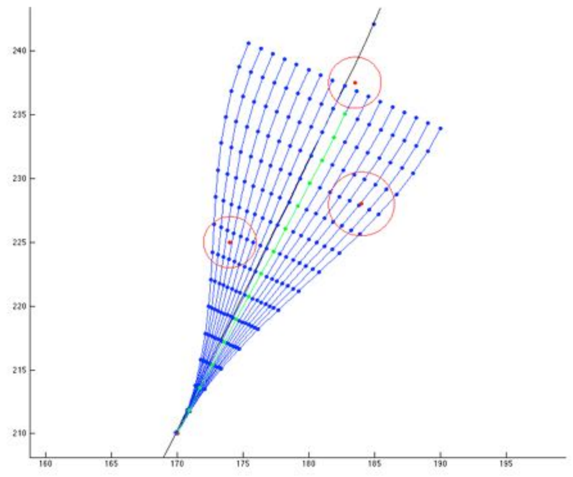


FIGURE 18 – NO COLLISION FREE PATH

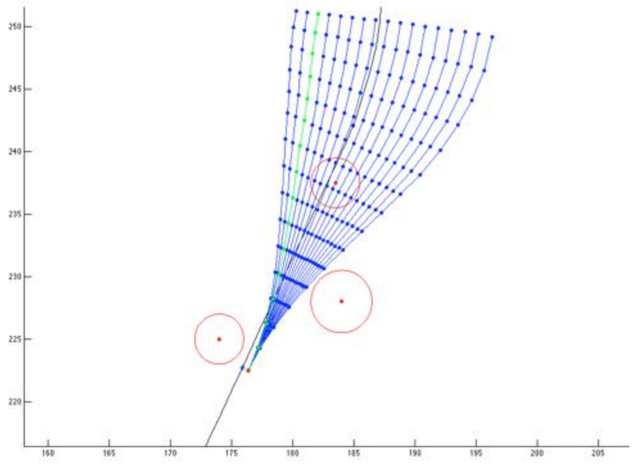


FIGURE 19 - COLLISION FREE PATH AVAILABLE

4.0 Testing

4.1 Simulation

Due to various factors outside of this projects control, testing the implementations functionality in the existing car code was initially carried out in a suitable simulator. The simulator used was Open Dynamics Engine (ODE), an open source, high performance rigid body simulator for Linux [20]. The world and vehicle used for simulation is shown in Figure 20 below.

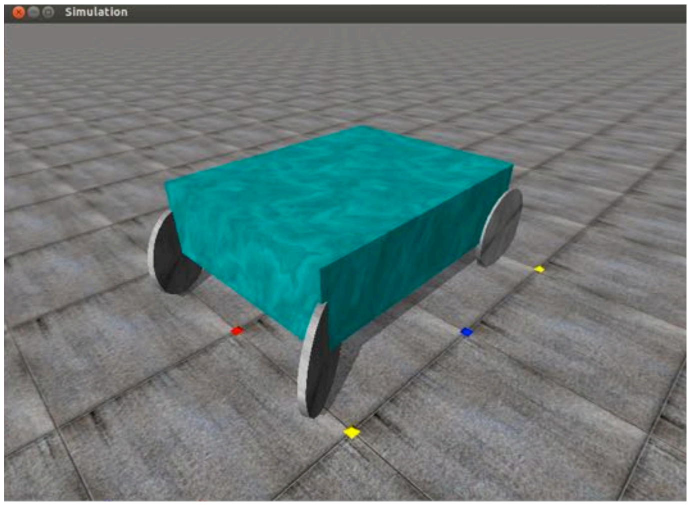


FIGURE 20 - SIMULATOR WORLD AND VEHICLE

4.1.1 Setup

Measurements of wheel distances were taken from the AutoSAE to best represent the vehicle and its turning capabilities in the simulator.

Longitudinal Wheel Distance: 1.8m
Lateral Wheel Distance: 1.4m

From last years drive by wire conversion project [13], the steering command – steering angle relationship was obtained to map between the control software and the simulator. The relationship can be expressed as:

$$\theta = 0.027\, x$$

Where θ is the steering angle in radians and x is the steering command from the control software. Using this relationship, the simulator can set the joint angle on the two font wheels providing similar behavior to the AutoSAE.

The existing car control code uses a GPS library for position, speed and heading information under normal operation [13]. The simulator provides body position and velocity in Cartesian (x, y and z) coordinates and body rotation with a 3x3 rotation matrix [21]. Consequently, with the exception of position, some conversion is required to bypass the GPS library and ensure the correct parameters (heading and speed) are passed to the car control code.

4.1.1.1 SPEED

The simulator provides the body velocity in x,y and z components. It is then trivial to calculate the vehicle speed by taking the magnitude of this vector.

$$speed = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

4.1.1.2 HEADING

A yaw angle is analogous to a heading and can be calculated from the body rotation matrix obtained from the simulator.

Given a rotation matrix returned from the simulator:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

yaw ϕ can be calculated as follows [22]:

$$\theta = -\sin^{-1}(r_{31})$$

$$\phi = \operatorname{atan2}\left(\frac{r_{21}}{\cos\theta}, \frac{r_{11}}{\cos\theta}\right)\frac{360}{2\pi}$$

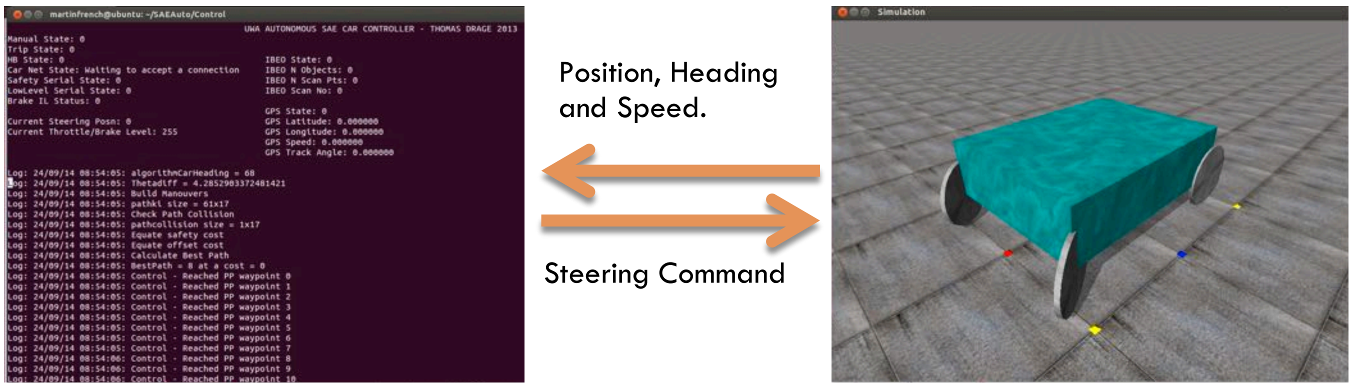
The atan2 function takes two arguments to obtain an angle in the correct quadrant between - π and π radians. As mentioned above, the existing car control code takes in a compass heading, which values from 0 to 360 degrees. Furthermore, in the simulator, a heading of 0 degrees aligns with the positive x-axis, while in the car code, 0 degrees aligns with positive y-axis. Hence an angle remap is required, which can be expressed as:

$$heading = \begin{cases} |\phi - 90|, & -180 \le \phi \le 90 \\ 360 - (\phi - 90), & otherwise \end{cases}$$

4.1.1.3 PROCESS COMMUNICATION

The calculated heading, speed and position can now be injected into the car control code, bypassing the GPS library.

The simulator is run as its own executable, separate from the car software. This was a choice made to keep the length of the car control code to a minimum and to ensure that the code remained manageable for others not needing the simulator. A disadvantage of this approach is that a transfer of bidirectional information must occur between the two processes. This pass of information is achieved using two Unix named pipes, each of which is a unidirectional FIFO buffer. Once each buffer is created, commands and feedback flow between the two programs as illustrated in Figure 21 below.


martinfrench@ubuntu: ~/SAEAuto/Control
Position, Heading and Speed.
Steering Command
Simulation

```
                                              UWA AUTONOMOUS SAE CAR CONTROLLER - THOMAS DRAGE 2013
Manual State: 0
Trip State: 0
HB State: 0                                         IBEO State: 0
Car Net State: Waiting to accept a connection       IBEO N Objects: 0
Safety Serial State: 0                              IBEO N Scan Pts: 0
LowLevel Serial State: 0                            IBEO Scan No: 0
Brake IL Status: 0
                                                    GPS State: 0
Current Steering Posn: 0                            GPS Latitude: 0.000000
Current Throttle/Brake Level: 255                   GPS Longitude: 0.000000
                                                    GPS Speed: 0.000000
                                                    GPS Track Angle: 0.000000

Log: 24/09/14 08:54:05: algorithmCarHeading = 68
Log: 24/09/14 08:54:05: Thetadiff = 4.2852983372481421
Log: 24/09/14 08:54:05: Build Manouvers
Log: 24/09/14 08:54:05: pathkl size = 61x17
Log: 24/09/14 08:54:05: Check Path Collision
Log: 24/09/14 08:54:05: pathcollision size = 1x17
Log: 24/09/14 08:54:05: Equate safety cost
Log: 24/09/14 08:54:05: Equate offset cost
Log: 24/09/14 08:54:05: Calculate Best Path
Log: 24/09/14 08:54:05: BestPath = 8 at a cost = 0
Log: 24/09/14 08:54:05: Control - Reached PP waypoint 0
Log: 24/09/14 08:54:05: Control - Reached PP waypoint 1
Log: 24/09/14 08:54:05: Control - Reached PP waypoint 2
Log: 24/09/14 08:54:05: Control - Reached PP waypoint 3
Log: 24/09/14 08:54:05: Control - Reached PP waypoint 4
Log: 24/09/14 08:54:05: Control - Reached PP waypoint 5
Log: 24/09/14 08:54:05: Control - Reached PP waypoint 6
Log: 24/09/14 08:54:05: Control - Reached PP waypoint 7
Log: 24/09/14 08:54:06: Control - Reached PP waypoint 8
Log: 24/09/14 08:54:06: Control - Reached PP waypoint 9
Log: 24/09/14 08:54:06: Control - Reached PP waypoint 10
```

FIGURE 21 - COMMAND AND FEEDBACK FLOW

Position, heading and speed are sent from the simulator to the car control software with a period of 100ms in the following format:

P,<px>,<py>,<pz>

B,<heading>

S,<speed>

A frequency of 100ms replicates the rate of information the fusion class provided in the existing car code [12]. The simulator is then providing a similar feedback environment found on the AutoSAE to the control code.

Similarly, a steering command is sent from the control software to the simulator every 50ms in the following format:

S,<command>

The AutoSAE can be run in manual acceleration mode to alleviate control issues presented by the slow GPS feedback and unresponsive drive system [12]. This manual mode is what is commonly used for practical testing so a design choice was made to exclude speed control in the simulator and revert to manual control.

4.1.2 Test Map

This test aims to verify the functionality of the implemented algorithm in the car control code within the simulation environment. The car in the simulator must travel around the defined waypoints and avoid obstacles along the way.

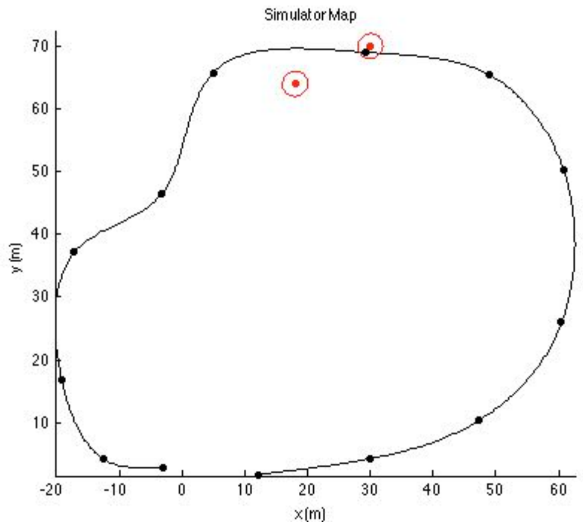


FIGURE 22 - SIMULATION TEST MAP

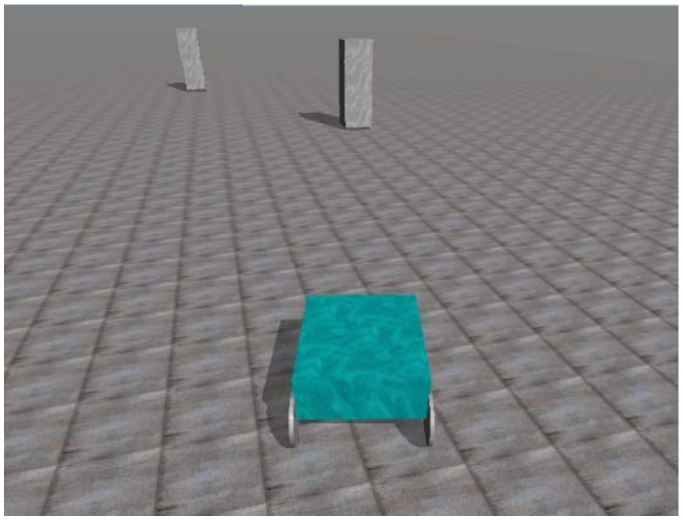


FIGURE 23 – OBSTACLES IN THE SIMLATOR

Figure 22 shows the map used in the simulator for this test. The black dots represent the defined waypoints, the black line is the generated base frame and the red dots are the two obstacles, shown in Figure 23. The locations of these two obstacles are known and hence were hard coded into the control software. Figure 22 also shows that one of the waypoints sits inside the footprint of the obstacle.

The test will firstly be run in the simulator at a manual constant speed, using the primitive path-planning algorithm implemented by Drage [12], and then using the advanced path-planning algorithm implemented in this project to illustrate the improvements in terms of obstacle avoidance.

4.1.3 Results

4.1.3.1 PRIMATIVE PATH PLANNING

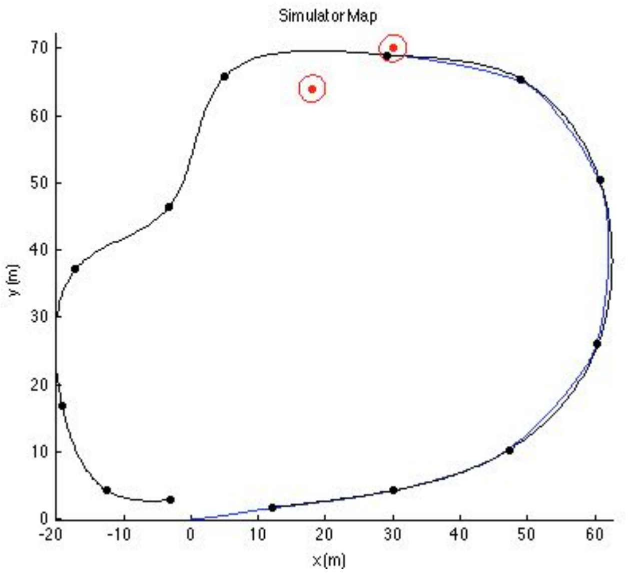


FIGURE 24 – PRIMATIVE PATH PLANNING MAP

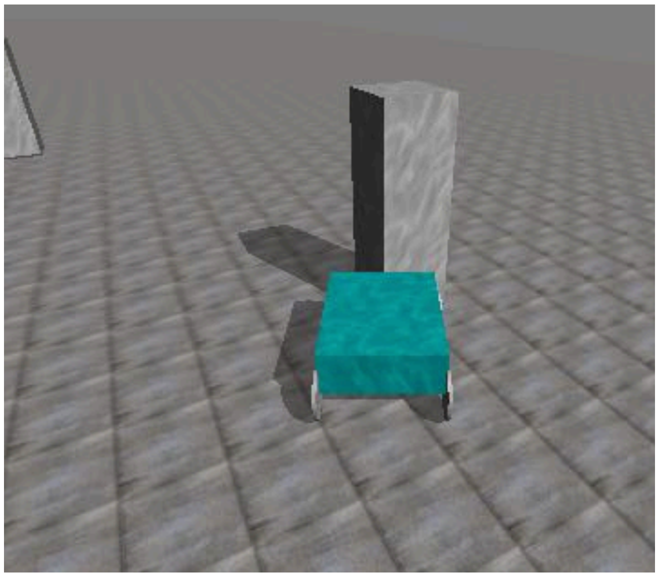


FIGURE 25 – PRIMATIVE PATH PLANNING COLLISION

In this simulation, the advanced path planning implemented in this project was disabled and the control code relied on the previously implemented waypoint-to-waypoint algorithm.

As a result, the vehicle collided with one of the obstacles, evident in Figure 25, and hence did not make it all the way to the final waypoint. Figure 24 shows the traveled path in blue, which clearly terminates at the first obstacle. This behavior is obviously undesirable and so this simulation provides a base point for improvement.

4.1.3.2 ADVANCED PATH PLANNING

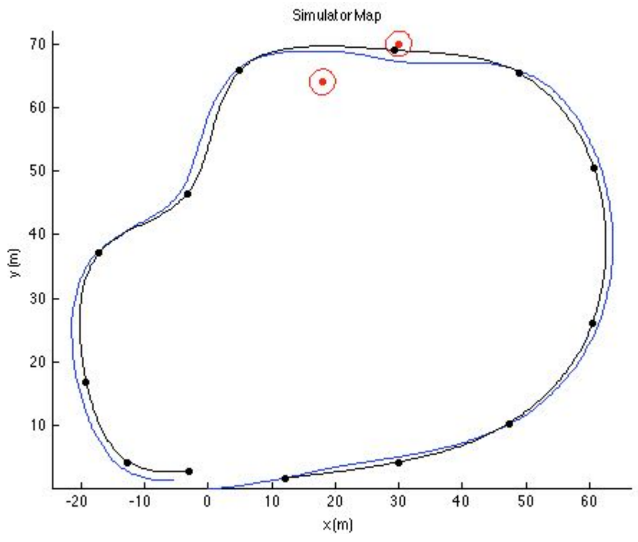


FIGURE 26 – ADVANCED PATH PLANNING MAP

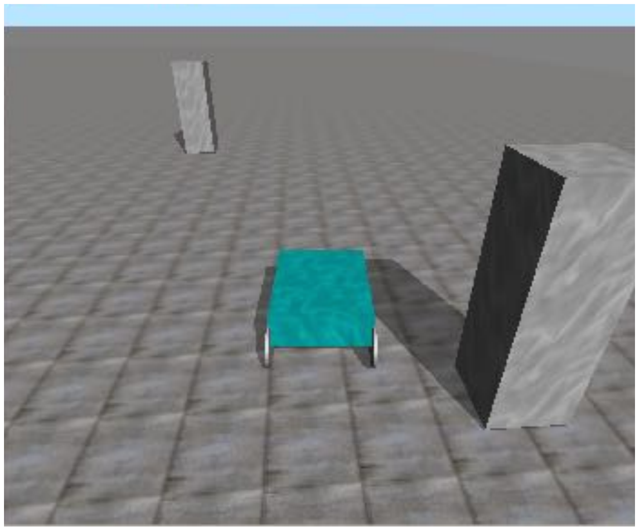


FIGURE 27 – ADVANCED PATH PLANNING OBSTACLE AVOIDANCE

In this simulation, the advanced path planning was re-enabled and the car was able to avoid the two obstacles as shown in Figure 27. Figure 26 shows a familiar map, with the traveled path again shown in blue. Evidently, the vehicle in the simulator using the implemented advanced path planning was able to travel the whole path.

A path-planning instance occurring in this simulation is shown in Figure 28 below. The chosen path is highlighted in green while the other blue paths are the other path candidates. Figures 29-31 show the various costs associated with this particular path plan, while Figure 32 shows a graph of the total cost which is the linear combination of the three cost functions as described in Section 3.5. The weightings used in this particular simulation were: 5.0, 0.5, and 0.1 for safety, offset and consistency costs respectively. Figure 32 also clearly shows path index 7 as the least cost path.

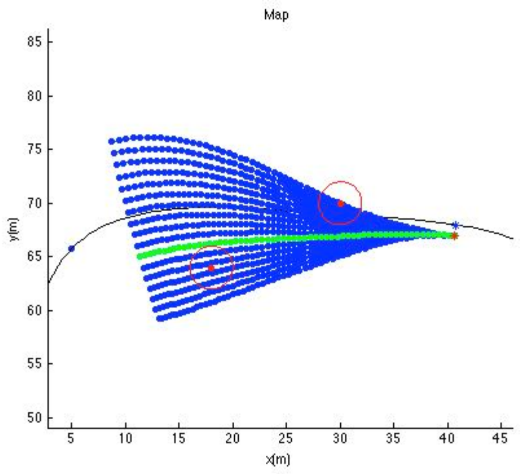


FIGURE 28 – NAVIGATING TWO OBSTACLES

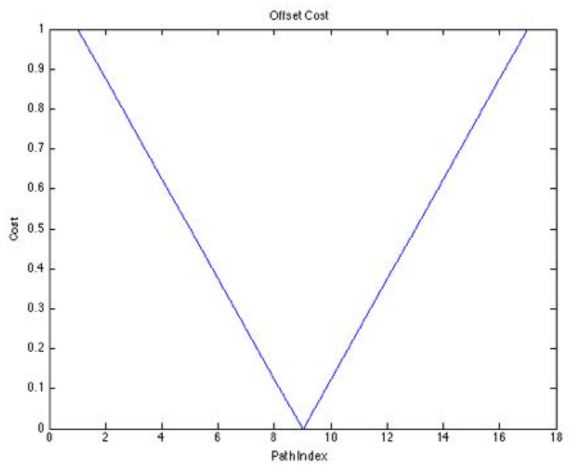


FIGURE 29 – GRAPH OF THE OFFSET COST

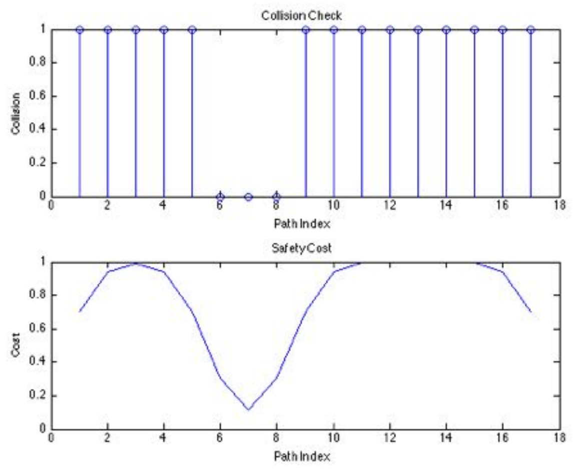


FIGURE 30 – BINARY GRAPH OF THE COLLISION CHECK AND A GRAPH OF THE SAFETY COST

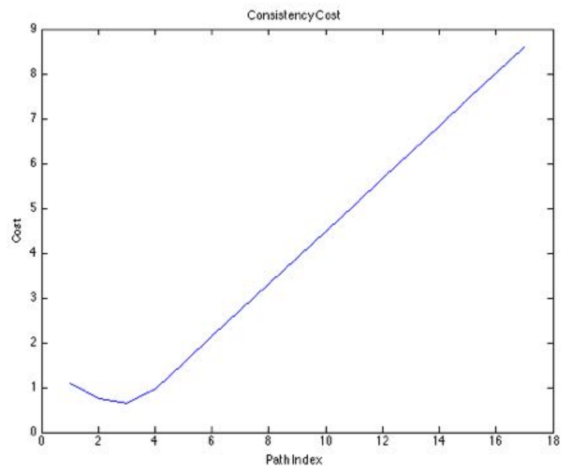


FIGURE 31 – GRAPH OF THE CONSISTENCY COST

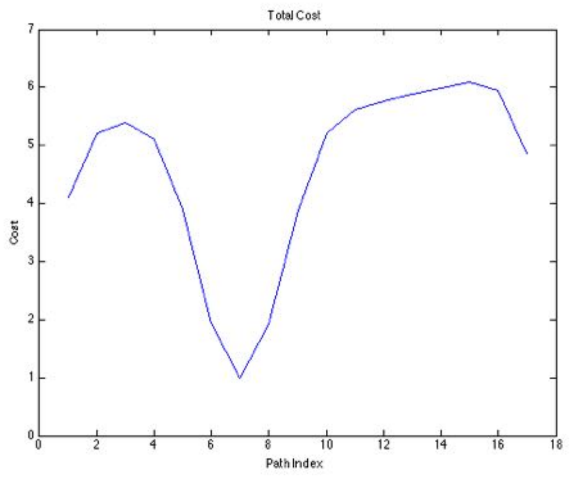


FIGURE 32 – GRAPH OF THE TOTAL COST

4.1.4 Discussion of Results

Clearly, the advanced path planning provides a significant safety improvement over the primitive implementation by successfully navigating around the two obstacles on the map. By comparing Figure 24 and Figure 26, it is clear that the advanced navigational algorithm does not stick to the base frame quite as accurately as the primitive algorithm. This is not a significant issue as the base frame is only there to provide directional information. The deviation is most likely due to the way in which the control software calculates it's bearing and steering command. Although run in a simulation, the car control code is running as it would on the AutoSAE and these results therefore prove correct functionality and implementation of the Advanced Navigational Algorithm into the existing car control code. The outcome of this test suggests that the implemented algorithm should work successfully on the AutoSAE, which is discussed in the following section.

4.2 Practical Testing

4.2.1 Test Map

Although the simulation proves the success of the algorithm implementation into the car code of the AutoSAE, it does not give any indication of its performance when used on the actual car. Testing was carried out within the limited space available at the UWA Sport Science oval next to the Business School. The map of waypoints and the generated base frame are shown in Figure 33.

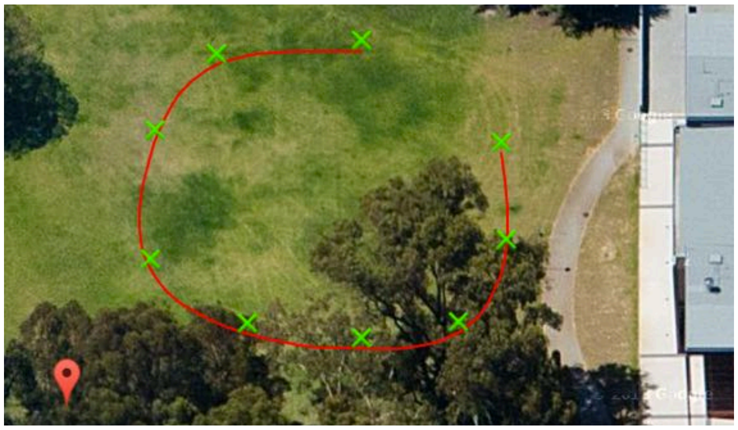


FIGURE 33 - PRACTICAL TESTING WAYPOINTS

4.2.2 Setup

Due to the small testing area and consequently tight set of waypoints, reduced manoeuvre parameters were set to ensure the car adequately stuck to the base frame. The manoeuvre parameters were set as follows:

Manoeuvre Length: 10m Max Lateral Offset: 3m Manoeuvre Granularity: 1m

These parameters produce a reduced set of 6 possible paths, like that shown in Figure 34, with reduced lengths of 10m compared to 30m which is what has been seen previously.

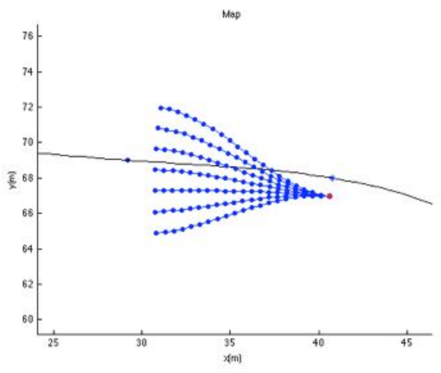


FIGURE 34 - REDUCED MANOEUVRE MAP

Unfortunately, the implementation of an accurate mapping a sensing system as described in Section 2.0, which included an accurate differential GPS system, and a map of detected upcoming obstacles was not fully implemented. This GPS system, which was planned for the AutoSAE in a parallel project, was found to be highly unreliable and consequently not fully implemented. Due to this unreliability, a change of scope occurred and as a result the obstacle detection was also never implemented.

This test was consequently run with the older, much less accurate GPS system without obstacle detection. The aim of this test is therefore to assess the implementations performance when inaccuracies of the GPS and the resulting effect on the car's steering control loop are taken into consideration.

Similar to the tests run in the simulator, the cars speed was manually controlled. In this test, a safety driver operating the throttle provided that speed control.

4.2.3 Results

The position results of three test runs are shown in Figure 35 where the black dots represent the defined waypoints and the colored lines represent the path travelled.

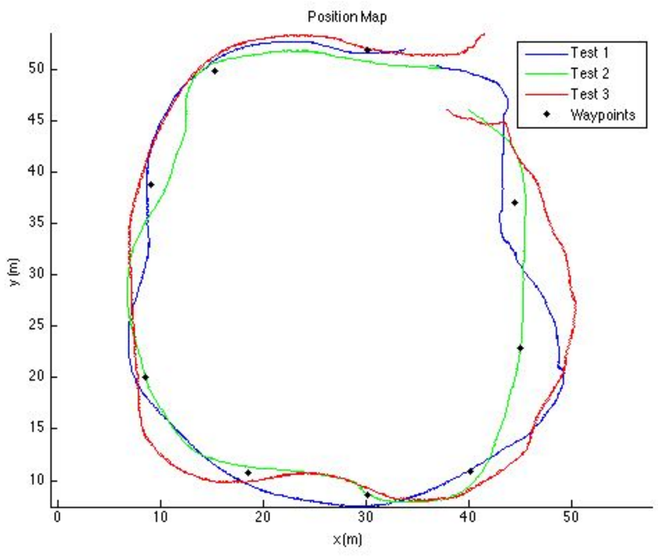


FIGURE 35 – POSITION RESULTS OF THREE TEST RUNS

4.2.4 Discussion of Results

Figure 35 shows the path traveled of three test runs on the same track. The inaccuracies of the path traveled compared to the base frame are unsurprising considering the results obtained by Drage [12] using the same control and GPS system. Despite these inaccuracies, this test proves that the implemented algorithm was successful in generating paths as the vehicle progressed around the track.

4.3 Recommendations for Future Work

4.3.1 Path Planning

This project provides a platform for further research in advanced path planning for the AutoSAE and autonomous racecars in general. Improvements and additional functionality can be implemented and tested on the foundations of this implementation by simply generating new cost functions to alter the path selection.

One such cost function is to look ahead on the base frame at the upcoming curvature to characterize where the vehicle should position itself to best approach a particular corner. This type of cost function will bring path planning for the AutoSAE into the racing domain where the vehicle is designed to operate. The goal would be to optimize the time the vehicle takes to traverse that corner or even the whole track. Before this cost function is implemented, the detection and characterization of the road edge must first be implemented.

The advanced path planning functionality could be extended to avoid not only static obstacles but also dynamic obstacles that might represent other cars on a racetrack.

A number of system limitations were discovered when testing the implemented algorithm in both the simulator and on the AutoSAE, these issues are the basis for the following recommendations.

Firstly, the AutoSAE experienced an issue in practical testing when traveling at slow speeds. The vehicle wouldn't follow the chosen path for long enough to complete a significant potion of the manoeuvre. This caused an issue because each manoeuvre has a section of path at the start where there is only a small change in offset. Reducing the manoeuvre length alleviated this issue, but is only suitable for constant speed testing, as the car would traverse the immediately planned path before a new plan was generated. A better solution would be to implement a speed dependent manoeuvre length, where the lengths of manoeuvres at a given path-planning instance change depending on the speed of the vehicle. This will ensure the vehicle can execute short manoeuvres, where the change in offset is fast (Figure 34) at slow speeds, while remaining stable with long manoeuvres, where the change in offset is slow (Figure 28) at high speeds.

A second issue arises when the vehicle control does not take the car along the base frame accurately enough resulting in missed waypoints. This becomes an issue because the vehicle control will want to turn the car around to visit this missed waypoint while the advanced path planning algorithm is only able take the vehicle one way along the base frame. A proposed solution, should the desired operation be to visit this missed waypoint, is to temporarily disable the advanced path planning, fall back on the primitive waypoint to waypoint method, and then finally re-enable advanced path planning to continue along the base frame once the waypoint has been hit.

4.3.2 AutoSAE

Throughout the project, difficulties with the hardware and safety systems of the AutoSAE were experienced at every step of the way. These difficulties motivate the following recommendations.

An accurate positioning system, as previously mentioned would provide a significant increase in the accuracy of the path-planning algorithm. Similarly, the vehicle control loops for steering and speed require significant tuning and debugging to ensure the vehicle is taken along the planned path with an increased level of accuracy.

Reliable obstacle detection using the LIDAR would enable real world testing of the advanced path-planning algorithm with obstacles.

One significant safety and reliability issue was the physical wiring to various sensors and the quality of core system hardware in the car. Although not in the scope of this project, physical rebuilds of two core system components were undertaken to significantly increase the reliability, safety and maintainability of the vehicle. It is recommended that implemented hardware in the future is built and installed to a satisfactory level of robustness and quality.

5.0 Conclusion

This project has focused on the implementation of local path planning for the UWA REV Autonomous SAE Electric Race Car (AutoSAE), amongst existing code in a practical way. The implementation and functionality of a suitable simulator, providing a quick way to test and improve implemented algorithms has additionally been presented. Using an algorithm tested and used by both Hanyang University in Korea and Harvard University in the US, this project was successfully able to implement an advanced path planning system, which was broken down into four sub tasks: base frame generation, candidate manoeuvre generation, manoeuvre cost calculation and manoeuvre selection.

This system was tested in both the simulator and on the AutoSAE. The results presented in section 4.1.3 demonstrate the implementations correct functionality in the both the simulator and on the AutoSAE itself. Limitations on the accuracy and performance of the algorithm were discovered during testing on the AutoSAE. These limitations were attributed to the inaccuracies of the positioning systems and vehicle control loops. It is therefore recommended that each of these systems receive some attention and tuning which will improve the performance of the implemented path-planning algorithm.

Scope for further work includes: the implementation of a speed dependent manoeuvre length, reducing the need for constant speed testing; implementing a new cost function, enabling the vehicle to position itself to best approach a particular corner, thus bringing the path planning algorithm into the racing domain and finally, extending the advanced path planning functionality to avoid not only static obstacles but also dynamic obstacles that might represent other cars on a racetrack.

Ultimately, this report has demonstrated that this project has successfully implemented an advanced path planning solution and provided a platform for further research in advanced path planning for the AutoSAE and the University of Western Australia.

References

[1] J. Schmidhuber, 'ROBOT CARS - autonomous vehicles - history of self-driving cars - best robot car', Idsia.ch, 2009. [Online]. Available: http://www.idsia.ch/~juergen/robotcars.html. [Accessed: May 17, 2014].

[2] ‘This Automobile Doesn't Need Driver’, *Palm Beach Daily News*, p. 4, December 15, 1966. [Online]. Available: Google News Archive, http://news.google.com/newspapers?id=MKskAAAAIBAJ&sjid=PaEFAAAAIBAJ&pg=3093,3999582. [Accessed: May 17, 2014].

[3] University of Washington, 'Park shuttle automated driverless vehicle pilot project - Netherlands', 2009. [Online]. Available: http://faculty.washington.edu/jbs/itrans/parkshut.htm. [Accessed: May 17, 2014].

[4] M. Maisto, 'Induct Now Selling Navia, First Self-Driving Commercial Vehicle', 2014.

[5] S. Blackstone, 'Watch An Audi A4 Weave Through Traffic Using Adaptive Cruise Control', 2012.

[6] D. Newcomb, 'Almost autonomous 2014 Mercedes-Benz E-Class', *MSN Autos*, 2013.

[7] K. L. R. Talvala, K. Kritayakirana, and J. C. Gerdes, "Pushing the limits: From lanekeeping to autonomous racing," *Annual Reviews in Control,* vol. 35, pp. 137-148, 2011.

[8] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel*, et al.*, "Stanley: The robot that won the DARPA Grand Challenge," *Journal of Field Robotics,* vol. 23, pp. 661-692, 2006.

[9] C. Keonyup, L. Minchae, and S. Myoungho, "Local Path Planning for Off-Road Autonomous Driving With Avoidance of Static Obstacles," Intelligent Transportation Systems, IEEE Transactions on, vol. 13, pp. 1599-1616, 2012.

[10] U. Ozguner, K. Redmill, and T. Acarman, Autonomous Ground Vehicles. Norwood: Artech House, 2011.

[11] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. P. How, and G. Fiore, "Real-Time Motion Planning With Applications to Autonomous Urban Driving," Control Systems Technology, IEEE Transactions on, vol. 17, pp. 1105-1118, 2009.

[12] T. Drage. “Development of a Navigation Control System for an Autonomous Formula SAE-Electric Race Car” BE thesis, UWA, 2013.

[13] J. Kalinowski. “Conversion of a Formula SAE Vehicle to Full Drive-by-Wire Capability” BE thesis, UWA, 2013.

[14] E. T. Y. Lee, "Choosing nodes in parametric curve interpolation," Comput. Aided Des., vol. 21, pp. 363-370, 1989.

[15] B. Guenter and R. Parent, “Computing the arc length of parametric curves,” IEEE Comput. Graph. Appl., vol. 10, no. 3, pp. 72–78, May 1990.

[16] H. Wang, J. Kearney, and K. Atkinson, “Arc-length parameterized spline curves for real-time simulation,” in Proc. 5th Int. Conf. Curves Surf., 2002, pp. 387–396.

[17] H. Wang, J. Kearney, and K. Atkinson, “Robust and efficient computation of the closest point on a spline curve,” in Proc. 5th Int. Conf. Curves Surf., 2002, pp. 397–406.

[18] M. W. Spong, S. Hutchinson, and M. Vidyasagar, Robot Modeling and Control. Hoboken, NJ: Wiley, 2006.

[19] T. Barfoot and C. Clark, “Motion planning for formations of mobile robots,” Robot. Auton. Syst., vol. 46, no. 2, pp. 65–78, Feb. 2004.

[20] R. Smith, 'Open Dynamics Engine - home', Ode.org, 2007. [Online]. Available: http://www.ode.org. [Accessed: Aug 10th, 2014].

[21] Ode-wiki.org, 'Manual: Data Types and Conventions - ODE Wiki', 2012. [Online]. Available: http://ode-wiki.org/wiki/index.php?title=Manual:_Data_Types_and_Conventions. [Accessed: Aug 10th, 2014].

[22] S. LaValle, 'Determining yaw, pitch, and roll from a rotation matrix', Planning.cs.uiuc.edu, 2012. [Online]. Available: http://planning.cs.uiuc.edu/node103.html. [Accessed: Aug 10th, 2014].

Appendix

5.1 Advanced Path Planning Function API

MATLAB Function Name	Inputs	Outputs	Description

`arclengthcurve`	`points`
	`Matrix of waypoints.`
	
	`granularity`
	`Granularity of base frame points.`
	
	`epsilon`
	`Acceptable numerical error.`
		`scoefx`
		`1x4 matrix Coefficients of arc length cubic spline x.`
		
		`scoefy`
		`1x4 matrix Coefficients of arc length cubic spline y.`
		
		`si`
		`Break points in the spline.`
			Generates an arc-length parameterized cubic spline (base frame) based on given waypoints.

`builddetailedbf`	`scoefx`
	
	`scoefy`
	
	`si`
	
	`granularity`
	`Granularity of the detailed base frame.`
		`sx`
		`X points.`
		
		`sy`
		`Y Points.`
		
		`ss`
		`Break points in the spline.`
			Generates a detailed (higher sampled) base frame for plotting and path generation.

`parevalspline`	`coefs`
	`The 4x1 coefficient matrix of the cubic spline.`
	
	`breaks`
	`Break points in the spline.`
	
	`t`
	`The point at which the cubic spline is`
		`result`
		`The result of the cubic spline expression.`
		
		`curvn`
		`The curve number of the spline where t resides.`
			Evaluate the derivative or value of a point along a cubic spline.

	to be evaluated. d Requested derivative [0,1,2].		
buildbfcurvature	dxds First derivative of x with respect to arc length. dyds First derivative of y with respect to arc length. dx2ds Second derivative of x with respect to arc length. dy2ds Second derivative of y with respect to arc length.	k The base frame curvature.	Calculate the base frame curvature at each point of s.
oblocalize	scoefx scoefy si ob A 3x? matrix describing the location and radius of ? obstacles. sguess An initial guess of spline segment to localize the obstacle. epsilon	arcob 2x? matrix detailing the position in arc length and the distance from the base frame.	Localize a set of obstacles onto the base frame with the curvilinear coordinate system.

`localize`	`coefx`
	
	`coefy`
	
	`breaks`
	
	`x0`
	`The x value of the point to localize.`
	
	`y0`
	`The y value of the point to localize.`
	
	`sguess`
	
	`epsilon`
		`value`
		`Arc length along the base frame.`
		
		`distance`
		`Distance from the base frame.`
		
		`loccurvn`
		`The spline curve number the point was localized on.`
		
		`count`
		`The number of iterations before a solution was found.`
			Localize a single point onto the base frame with the curvilinear coordinate system.

`evalheading`	`scp`
	`The arc length of the base frame to calculate heading at.`
	
	`ss`
	
	`dxds`
	
	`dyds`
		`sindex`
		`The spline curve number the arc length was found on.`
		
		`paththeta`
		`The heading at that arc-length on the base frame.`
			Find the heading of a point on the base frame represented by its arc-length.

`buildmanouvers`	`scp`
	`The arc length of the base frame to build manoeuvres from.`
	
	`cpdistance`
	`Distance from the base frame.`
	
	`px`
	`X value of point to build manoeuvres`
		`mxi`
		`Matrix of i manoeuvres x points.`
		
		`myi`
		`Matrix of i manoeuvres y points.`
		
		`pathki`
		`Matrix of i manoeuvres path`
			Generate a set of i manoeuvres from a point.

	from. py Y value of point to build manoeuvres from. maxlatoffset Maximum lateral offset. mangran Manoeuvre granularity. (Distance between manoeuvres) manlength Manoeuvre Length. mincurverad Minimum curvature radius to allow. thetadiff Difference in vehicle heading and base frame heading. ss sindex kk The base frame curvature. paththeta Base frame heading.	curvature. pathqi Matrix of i manoeuvres path offsets. dthetai Matrix of i manoeuvres derivative of theta. mans The arc lengths of the set of manoeuvres.	
checkpathcollision	arcob pathqi s The arc lengths of	pathcollision The collision matrix.	Generate a collision matrix for the set of given manoeuvres.

	the set of manoeuvres.		
equatesafetycost	mangran pathcollision	safetycost The safety cost for each manoeuvre i.	Calculate the safety cost of each given manoeuvre.
equateoffsetcost	pathqi	pathoffsetcost The offset cost for each manoeuvre i.	Calculate the offset cost of each given manoeuvre.
equateconscost	previouspathq The previous path planning instances path offset and arc-length matrix. pathqi mans	pathconscost The consistency cost for each manoeuvre i.	Calculate the consistency cost of each given manoeuvre.
mincost	Ks Safety weighting. safetycost Kpo Path offset weighting. pathoffsetcost Kc Path consistency weighting. conscost	totalcosts The total cost of each path i. bestpath The path index associated with the least cost.	Calculate the total cost of each path i and the best path index after weighting.
genprevpathq	bestpath pathqi mans	previouspathq	Store the best paths offset and arc-length into a matrix for the next path-planning instance.