Racing Line Optimization

by

Ying Xiong

B.E., Computer Science

Shanghai Jiao Tong University (2009)

Submitted to the School of Engineering

in partial fulfillment of the requirements for the degree of

Master of Science in Computation for Design and Optimization

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

Signature of Author . . . . . .

Computation for Design and Optimization, School of Engineering

July 30, 2010

Certified by . . . . . .

Gilbert Strang

Professor of Mathematics

Thesis Advisor

Accepted by . . . . . .

Karen Willcox

Associate Professor of Aeronautics and Astronautics

Co-Director, Computation for Design and Optimization Program

Contents

Abstract

Although most racers are good at controlling their cars, world champions are always talented at choosing the right racing line while others mostly fail to do that. Optimal racing line selection is a critical problem in car racing. However, currently it is strongly based on the intuition of experienced racers after they conduct repeated real-time experiments. It will be very useful to have a method which can generate the optimal racing line based on the given racing track and the car. This paper explains four methods to generate optimal racing lines: the Euler spiral method, artificial intelligence method, nonlinear programming solver method and integrated method. Firstly we study the problem and obtain the objective functions and constraints for both 2-D and 3-D situations. The mathematical and physical features of the racing tracks are studied. Then we try different ways of solving this complicated nonlinear programming problem. The Euler spiral method generates Euler spiral curve turns at corners and it gives optimal results fast and accurately for 2-D corners with no banking. The nonlinear programming solver method is based on the MINOS solver on AMPL and the MATLAB Optimization Toolbox and it only needs the input of the objective function and constraints. A heavy emphasis is placed on the artificial intelligence method. It works well for any 2-D or 3-D track shapes. It uses intelligent algorithms including branch-cutting and forward-looking to give optimal racing lines for both 2-D and 3-D tracks. And the integrated method combines methods and their advantages so that it is fast and practical for all situations. Different methods are compared, and their evolutions towards the optimum are described in detail. Convenient display software is developed to show the tracks and racing lines for observation. The approach to finding optimal racing lines for cars will be also helpful for finding optimal racing lines for bicycle racing, ice skating and skiing.

1. Introduction

In racing sports, the racing line is the route that the vehicle takes. For a given track, there are an infinite number of racing lines possible. An optimal racing line minimizes the time needed to complete the course. A comparison of two possible racing lines for the same racing track is shown in Figure 1.1.

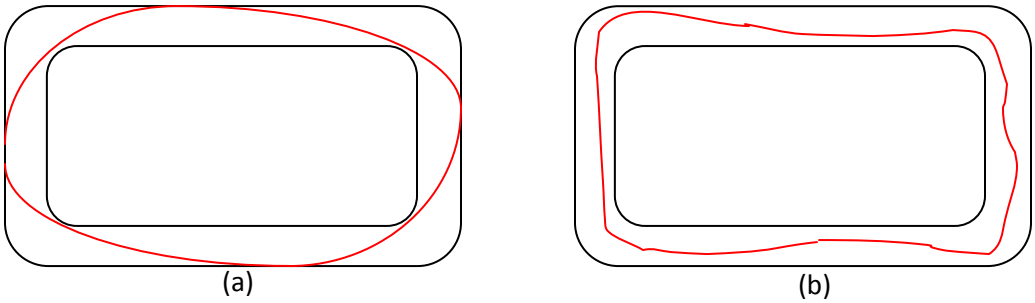


Figure 1.1. Two different racing lines on the same track. (a) makes use of the turns and obtains a smooth and consistent racing line, while (b) has random walks and unnecessary small turns which will lower the speed and take more time.

An optimal line considers the conditions of the track and makes smart decisions based on the track. For example, comparing the two racing lines in Figure 1.1 for the same track, (b) is making many random unnecessary turns, while (a) wisely avoids unnecessary turns and makes the line smoother. So (a) is strategically wiser than (b). However, it may not be the best solution.

It is obvious that in racing games corners make a large difference in performance. On straight tracks, theoretically all racers can reach the maximum speed possible and just go in a straight line; thus there is not much difference for racers' different skills. But when there is a turn, the speed cannot go above the allowed level, and there is a trade-off between the speed and the length of racing line taken. A smoother racing line with smaller curvature is longer, and a more curvy racing line may be shorter. Statistics show that successful car racing champions are always following the optimal racing line while other racers are constantly not getting the optimal line. [1] The slowest part of the racing track differentiates good and bad racing techniques.

Suppose a_n is the centripetal acceleration, and $a_n = \frac{v^2}{r}$. When a_n is a fixed number, we have that v^2 is proportional to r, i.e. $v^2 \propto \frac{1}{k}$ where v is the maximum speed allowed and r is

the radius of the corner. When r increases, v will increase. The larger r is, the less control it has over the speed.

Actually, when r is infinitely large, the corner becomes a straight line, and the maximum speed allowed will just be the physical limit of the car $v_{\max}$ (Figure 1.2). In contrast, consider the curvature k. When k increases, the maximum speed allowed decreases (Figure 1.3).

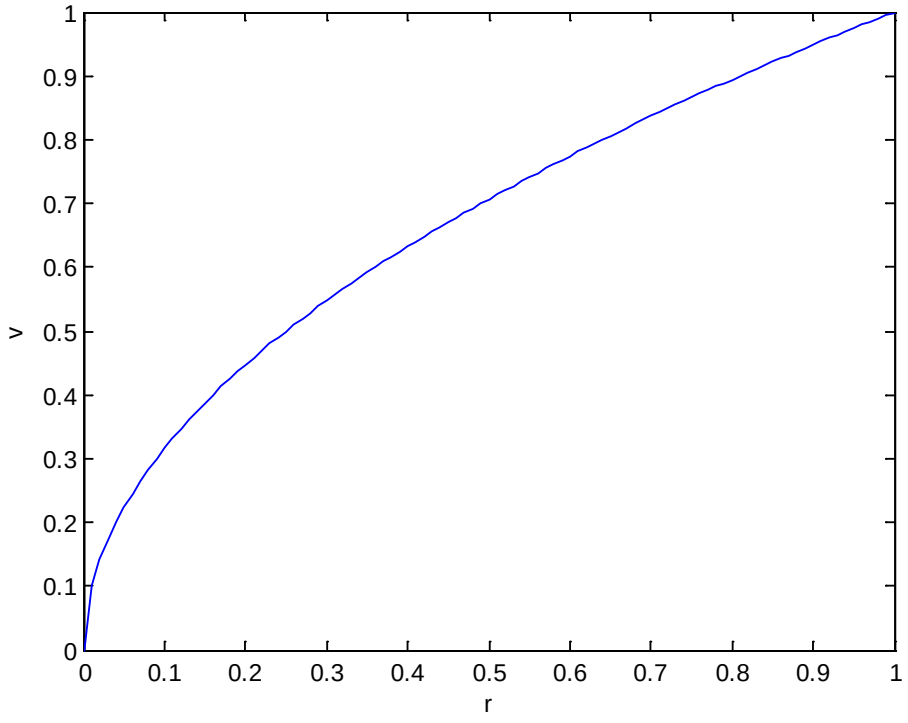


Figure 1.2. Relationship between the maximum allowable speed v and the radius of the corner r Maximum speed goes up when radius goes up.

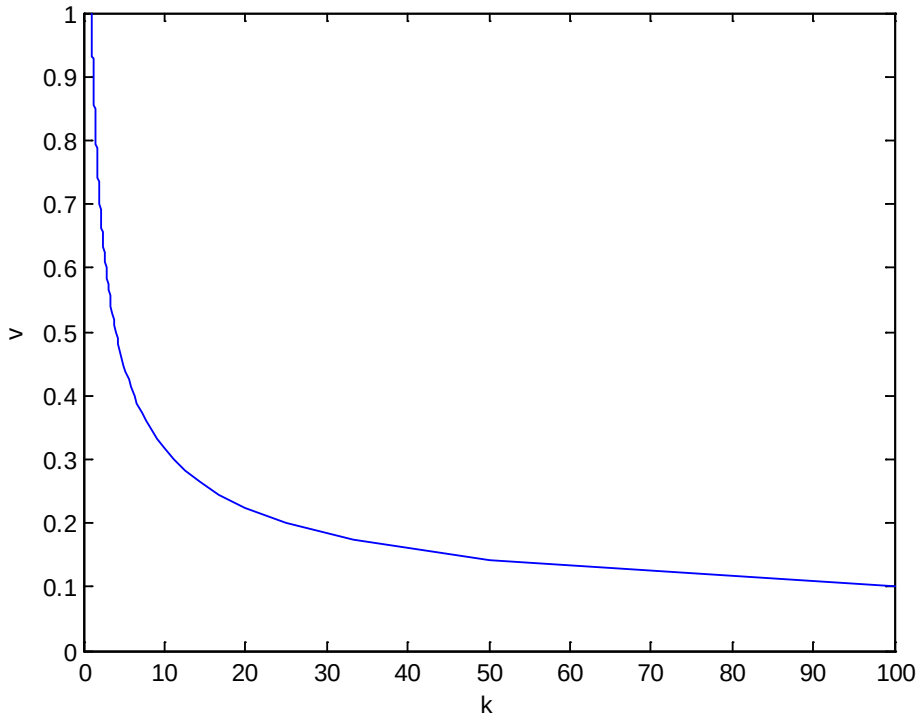


Figure 1.3. Relationship between the maximum allowable speed v and the curvature of the track k. Maximum speed goes down when curvature goes up.

The racing line at corners depends on the following factors: braking point, turn in point, apex and the position and direction of the next corner.[19]

Let's start with the problem of only going through one corner. When analyzing a single corner, the optimal line is the one that minimizes the time cost during the corner and maximizes the overall speed of the vehicle through the corner. If one uses the path with the smallest radius, the distance travelled around the corner is minimized. However, by fitting a curve with a wider radius, i.e., smaller curvature, into the corner, higher speeds can be maintained. This may compensate for the extra distance travelled. When analyzing the whole track, the optimal racing line minimizes the total time and maximizes the overall speed around the track.

Some research has been done on this topic to find optimal racing lines. However, much of it focuses on highly simplified physical conditions and does not have a complete analysis of the real situation. There is also no work regarding 3-dimensional tracks, which are actually the most commonly seen racing tracks, and the 3-dimensional condition is much more complicated. It has

some major differences with the 2-dimensional track. And in published papers, the optimization objectives are all set to be the minimization of integral of curvature squared $\int k^2 ds$, a commonly seen expression for calculating the elastic potential energy, without clear explanation of why k^2 is used here; or directly the time lapse which will depend heavily on how the speed function is correlated with the racing line. We will also show that the integral of the square root of k ($\int \sqrt{k} ds$) instead of $\int k^2 ds$ is often a better optimization objective by experimental results. And the minimization of $\int k^2 ds$ is practically achieved by the Euler spiral for one corner. Thus the Euler spiral is *not* guaranteed to be the best solution, just one of the close-to-best solutions.

2. Problem formulation

The objective of the problem is to minimize the time cost for the car to complete the whole racing track.

$$t = \int dt = \int \frac{dt}{ds} ds = \int \frac{1}{v} ds$$

where t is the total time cost, s is the length that the car travels through, and v is the velocity of the car.

So the time cost is the integral of the reciprocal of speed from zero to the total length travelled.

There are several constraint conditions with respect to the road (track) condition and the car features. We will study the optimization problem of minimizing t with various constraints. We will first analyze the 2-dimensional racing track, and then extend it to the more general 3-dimensional racing track.

Figure 2.1 shows two screenshots in two electronic racing games. The track in (a) is a 2-dimensional track while the track in (b) is a 3-dimensional track. (b) looks more like a real life simulation with the complicated road condition, different bending angles and banked turns.

(a) (b)

Figure 2.1. Demonstration of 2-D racing track and 3-D racing track by video game screenshots. (a) is from a 2-D racing game and (b) is from a 3-D racing game.

Clearly, in Figure 2.1 (a) the track can be represented by a 2-D geometric shape, like the view in the figure that is looking directly down. However, the track in Figure 2.1(b) is a lot more complicated as the car bends toward the right side and its left wheels are lower than its right wheels due to the banked corner.

2.1 Problem formulation for two-dimensional racing tracks

Let's start with the two-dimensional (2-D) racing tracks. Two-dimensional racing tracks are flat tracks with no slopes or banked corners along the way. In an x-y-z space the whole track can be expressed with z=0.

There are some constraints for a car on a 2-D track. We will analyze them one by one.

1) Assume that there is no skid. The car is always running within control. To satisfy the non-skid constraint, the velocity should not be too large at a turn, so that the friction force on the car can provide the centripetal acceleration needed (Figure 2.2).

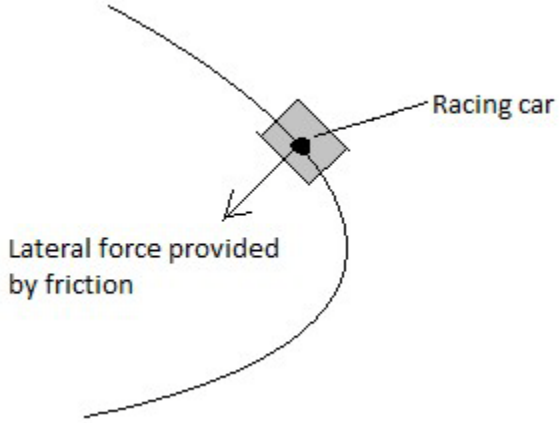


Figure 2.2. On a 2D track, the lateral force to support the turning of the car is provided by the friction force only.

$$m\frac{v^2}{r} \le \mu mg \iff \frac{v^2}{r} - \mu g \le 0 \; i.e. \; kv^2 - \mu g \le 0.$$

Here m is the mass of the car, μ is the friction coefficient, and g is the acceleration of gravity.

The value of μ depends on the road conditions and the car's features. Ignoring the difference between car wheels, a table of the friction coefficients that are usually used is in Appendix (A).

2) The car moves an angle of $\Delta\theta$ degrees in total (Figure 2.3). From the formula of curve length, we know that

$$\int \frac{1}{r} ds = \Delta\theta \;\; i.e. \;\; \int k ds = \Delta\theta$$

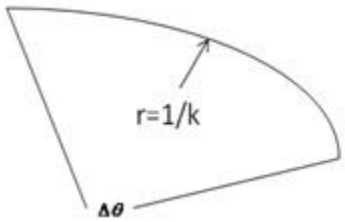
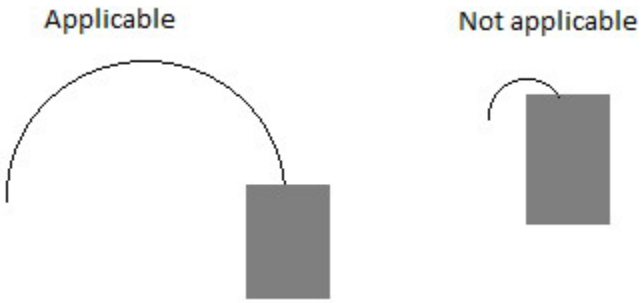


Figure 2.3. $\triangle\ \theta$ can parameterize the distance that the car has travelled along the line, given the radius information at all the points and that the track that bounds a convex region.

3) There is a physical limit for cars to turn. For example, a car cannot turn immediately 180 degrees. This indicates that the turning radius of the car cannot be too small (Figure 2.4).

$$r \geq r_{min}\ i.e. k \leq k_{max}$$



(a) (b)

Figure 2.4. The turning of a car with the grey rectangle representing a car. (a) is an applicable turn because the radius of the turn is large enough. But (b) is not applicable because the radius is too small and the car cannot turn so dramatically at once.

4) Speed limit by the engine or by racing regulations. The car cannot run with infinitely large speed. There is a maximum speed limit v_{max} .

$$v \leq v_{max}$$

5) Acceleration limit by engine. There are upper and lowers bounds for the allowable acceleration

$$a_{min} \leq a \leq a_{max}$$

a_{min} is a negative number and defines the maximum deceleration; a_{max} is a positive number and defines the maximum acceleration. From the F1 car features, we set $a_{min} = -4g$, $a_{max} = 1.45g$ (referring to Appendix (B)). It is obvious that the absolute

value of deceleration is much larger than the absolute value of acceleration. This ensures the safety of the racer.

Summarizing the constraints, the optimization problem is formulated as follows:

$$\text{Minimize } \int \frac{1}{v} ds$$

$$\begin{aligned} s.t. \quad & kv^2 - \mu g \le 0 \\ & \int k ds = \Delta\theta \\ & k \le k_{\max} \\ & v \le v_{\max} \\ & a \le a_{\max} \\ & a \ge a_{\min} \end{aligned}$$

This is a nonlinear optimization problem involving dynamic programming. The objective function is the total time cost expressed as an integral of a function of distance travelled. There are six constraints. Two of them are nonlinear constraints and four of them are linear. When the number of variables is large, it will be a large-scale optimization problem.

2.2 Problem formulation for three-dimensional tracks

Three-dimensional (3-D) tracks are the most commonly seen tracks in real life racing. Three-dimensional refers to both the biased banking of the track to the left or right and the up and down slopes of the track. Different tracks have different 3-D features. Some racing tracks like NASCAR tracks have more banked corners, while some racing tracks like Formula One tracks have less banking but more up and down slopes.

The basic concepts of dealing with the two-dimensional and three-dimensional tracks are similar, but the three-dimension situation is much more complicated and the force analysis is very different.

We will first do force analysis, give the 3-dimensional problem formulation and then introduce the way of representing the tracks in 3-D.

2.2.1 Force analysis

It makes a difference whether we consider the car as a single point or as an extended object. If we see the car as an extended object, its two wheels may receive different forces. If we see the car as only one point, all forces are applied to its geometric center of gravity.

In the roller-coaster track in Figure 2.5, assume that the car is undergoing circular movement in the vertical plane with a constant speed v.

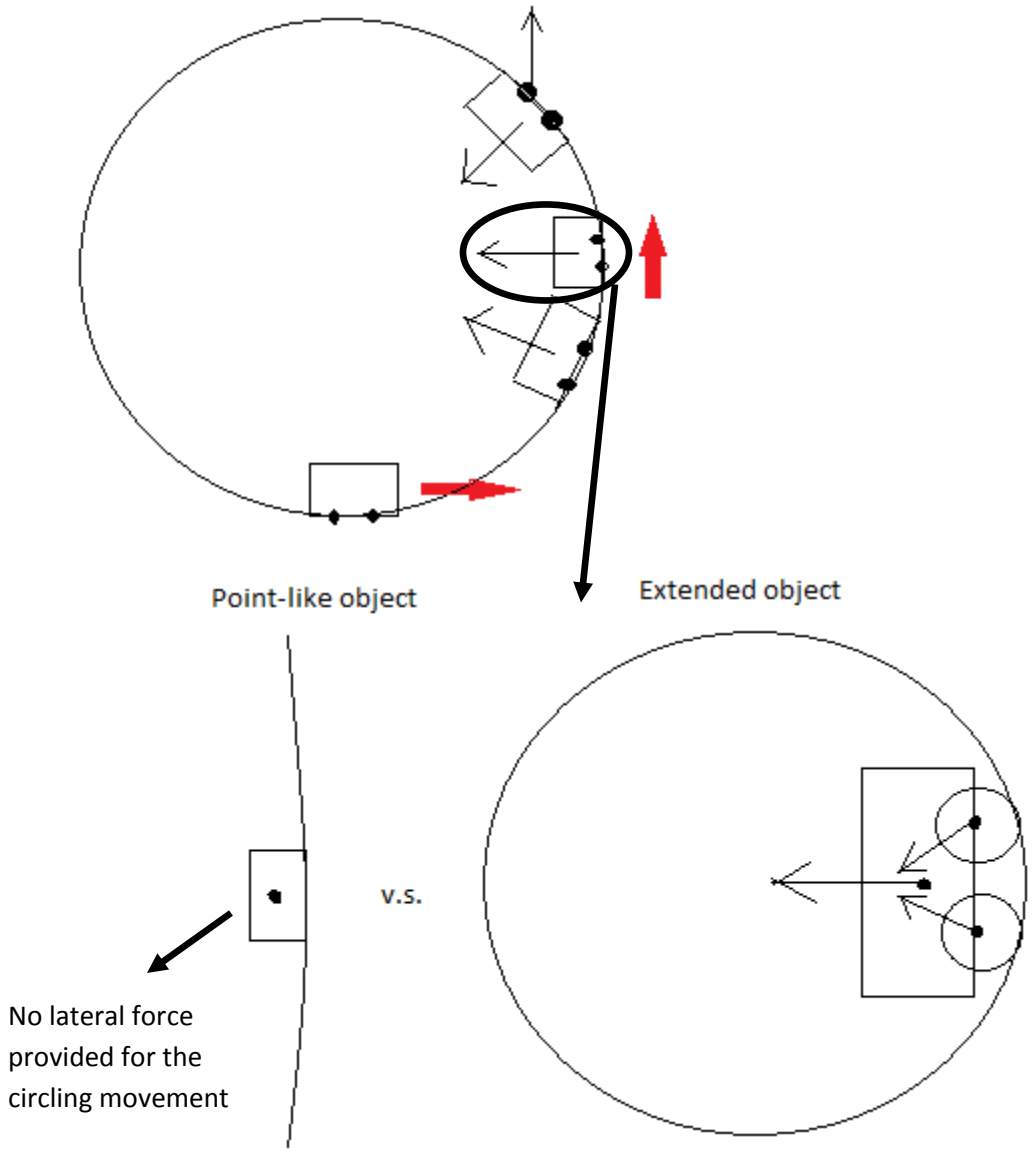


Figure 2.5. Roller-coaster track that can be seen in acrobatics. When the car is at the 3 o'clock position, the force analysis is different when we consider the car as a point and when we consider the car as an extended object. But usually we can ignore the difference.

If we see the car as a single point and assume no extra down force on the car, there is no force that can serve as a centripetal force to sustain the cornering, and the car is supposed to fall down at this point. However, if we look at the car as an extended object, the two wheels are receiving some forces to make the turning possible.

In the analysis of this thesis, the car is regarded as a point instead of an extended object. The difference is ignored because in our racing track, the value of $\left|\frac{d\vec{N}}{ds}\right|$ is very small. This means that the plane of the road is not changing too quickly – the problem described in the "roller coaster" track will not occur.

The force graphs which consider the car as a single point object are in Figures 2.6 and 2.7.

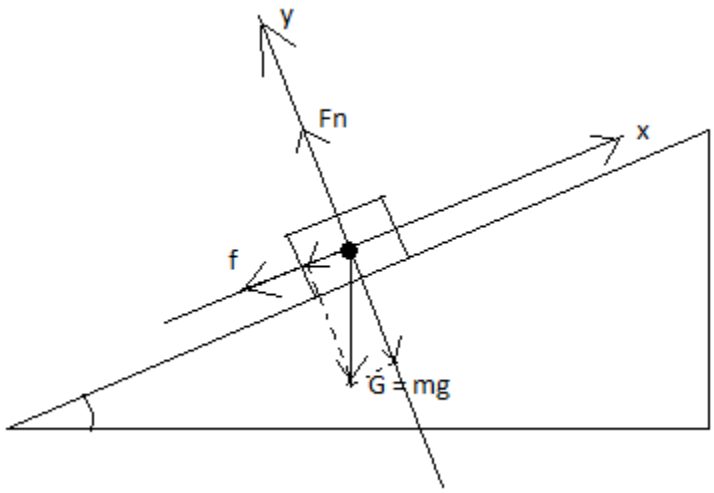


Figure 2.6. Force analysis of the racing car on the cutting plane orthogonal to the direction of the racing line when it is considered as a single point.

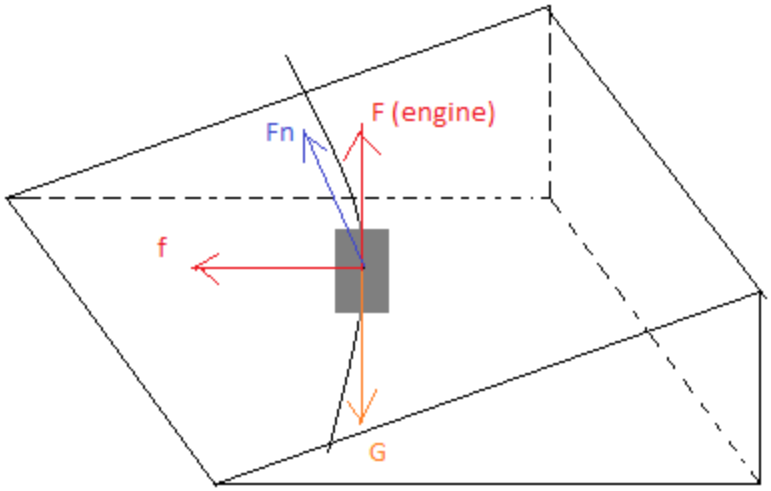


Figure 2.7. Three-dimensional view of the force analysis of the racing car neglecting the rolling friction and air drag forces

1) Firstly, let's not consider the up and down slopes. So the center line of the track is always on the same horizontal plane, and it is only that the road is leaning to left/right at times. This is a practical assumption, especially for racing cars like NASCAR. Tracks only have some leaning at cornering parts to help racers get higher speeds.

   Consider the forces along the lateral and tangential directions. Here, α is the angle between the direction of the car's motion and the center line. θ is the angle of the banking.

   Lateral direction:

$$\pm mg\sin\theta\cos\alpha + \mu mg\cos\theta$$

   Tangential direction:

$$F_{car} \pm mg\sin\theta\sin\alpha$$

   Here the sign $\pm$ in lateral force depends on the way the racing line curves – inwards or outwards. The sign $\pm$ in tangential force depends on whether the tangential line of the racing line is on the left or right of the center line. Compare it with the 2-dimensional situation:

   Lateral direction: μmg

   Tangential direction: F_{car}

The constraint is firstly on the lateral part. $\mu mg \le m\frac{v^2}{r} \Rightarrow v \le \sqrt{\frac{\mu g}{k}}$. Then the tangential part will constrain the acceleration to be applicable.

Back to the 3-dimensional model, similarly, we have

$$\pm mg\sin\theta\cos\alpha + \mu mg\cos\theta \ge m\frac{v^2}{r} \Rightarrow v \le \sqrt{\frac{(\mu\cos\theta \pm \sin\theta\cos\alpha)g}{k}}$$

And when the $\pm$ is chosen as minus, $\tan\theta\cos\alpha \le \mu$ must be satisfied, which requires

$$\cos\alpha \le \frac{\mu}{\tan\theta}$$

2) For the full three-dimensional model, assume that there are actually up and down slopes. The following forces are considered: gravity, support force, lateral friction and pull force by the car engine. Here we neglect all the other forces.

$$\vec{F} = \vec{G} + \vec{F}_N + \vec{f} + \vec{F}_{car}$$

where $\vec{F}$ is the total force that the car receives.

$$\vec{F} - \vec{G} = \vec{F}_N + \vec{f} + \vec{F}_{car}$$

$$\Rightarrow \frac{\left|\vec{F} - \vec{G}\right|}{m} = \frac{\left|\vec{F}_N + \vec{f} + \vec{F}_{car}\right|}{m}$$

$$\Rightarrow \left|(\vec{a} - \vec{g}_1)\cdot\vec{n}_{car}\right| = \left|\frac{1}{m}\vec{f}\cdot\vec{n}_{car}\right| \le \mu g\cos\theta$$

$$\Rightarrow a_{\min} \le (\vec{a} - \vec{g}_1)\cdot\vec{n}_{car} \le a_{\max}$$

$\vec{n}_{car}$ is the direction orthogonal to the direction of the motion of the car in the plane of the road.

3) The model we have in 2) is not complete. It does not consider three other factors.

   a) Rolling friction $\vec{f}_{roll}$.

      Rolling friction is usually negligible compared to kinetic friction, but it still counts. An obvious example is that when you are driving and you stop the engine, the car will actually slow down and stop gradually. The whole stop procedure by rolling friction and

air drag force does not take too much time. So rolling friction is worth considering. The value of rolling friction coefficient is usually around 0.001.

b) Drag force $\vec{F}_{dr}$

In fluid dynamics, drag (sometimes called air resistance or fluid resistance) refers to forces that oppose the relative motion of an object through a fluid (liquid or gas). Drag forces act in a direction opposite to the oncoming flow velocity. Unlike other resistive forces such as dry friction, which is nearly independent of velocity, drag forces depend on velocity. When racing cars are moving at a very high speed in the air, the drag force of the air may become significant. The formula for drag force is $F_D = \frac{1}{2}\rho v^2 C_d A$, where ρ is the density of the air, v is the velocity relative to the air, A is the reference area, and C_d is the drag coefficient. It is a dimensionless parameter and it is usually 0.25 to 0.45 for a normal car and 0.8 to 1.1 for F1 racing cars. F1 cars have higher drag coefficient, but they have comparatively small reference area as well.

c) Down force and lift force $\vec{F}_L$

The down force of the car is often called "ground effect", because cars have an extra force pushing down to achieve higher speeds at corners. The down force comes from air pushing on the wings of the car or from the low pressure created beneath cars with special designs. The formula for down forces is similar to the formula for drag forces: $F_L = \frac{1}{2}\rho v^2 C_L A$. The existence of down force may significantly change the value of friction forces and change the strategy of racing as well. Lift force refers to the force that is lifting the car when the speed is very large.

So with all factors considered, the total force that the car receives is

$$\vec{F} = m\vec{a} = \vec{G} + \vec{F}_N + \vec{f} + \vec{F}_{car} + \vec{F}_{dr} + \vec{f}_{roll} + \vec{F}_L$$

$\vec{G}$ is the gravitational force on the car which is pointing toward the ground.

$\vec{F}_N$ is the normal force from the road which is perpendicular to the racing track.

$\vec{f}$ is the friction force which is parallel to the plane of the track but orthogonal to the motion of the car.

$\vec{F}_{car}$ is the pulling force of the car which is assumed to be parallel to the motion of the car.

$\vec{F}_{dr}$ is the drag force of the car, i.e. air resistance, which opposes the relative motion of the car through the air.

$\vec{f}_{roll}$ is the rolling friction which is assumed to be parallel and just opposite to the motion of the car.

$\vec{F}_L$ is the down force and lift force, which is perpendicular to the racing track.

2.2.2 Three-dimensional constraints

Ground effect (down force) used to be a very popular technique in racing in the 20th century. Indy cars still employ ground effect to some extent [12], but recently it has been restricted or forbidden in some racing events especially in F1 due to safety concerns [2]. If the lateral friction of the car largely relies on the huge down force instead of the gravitational force, it can be very dangerous when the down force suddenly disappears due to technical error or malfunction of the car – the car will skip away and the racer will be in danger. Lift force from the air is usually very small and negligible compared to down force. To simplify the problem, we assume that no down force is allowed and no lift force is significant enough to be considered. i.e. $\vec{F}_L = 0$. And comparatively, drag force always exists and we are going to take it into account. The data found for $\vec{F}_{dr}$ is: air density $\rho \approx 1.2 kg/m^3$, and $C_d A \approx 0.6 m^2$. [4]

$\vec{n}$ is the normal vector of the surface of the track, $\vec{t}_{car}$ is the tangential vector of the racing line the car takes, and $\vec{n}_{car}$ is the normal vector of the racing line in the plane of the road (pointing to the right). The force components in the three directions are:

i) $\left(\vec{F} - \vec{G} - \vec{F}_L\right) \cdot \vec{n} = \left|\vec{F}_N\right|$

ii) $\left(\vec{F} - \vec{G} - \vec{F}_L\right) \cdot \vec{t}_{car} = \vec{F}_{car} \cdot \vec{t}_{car} - \left|\vec{F}_{dr}\right| - \left|\vec{f}_{rolling}\right|$

iii) $\left(\vec{F} - \vec{G} - \vec{F}_L\right) \cdot \vec{n}_{car} = \left|\vec{f}\right|$

Note that

$$\vec{F}_L \cdot \vec{n}_{car} = 0,\ \vec{F}_L \cdot \vec{t}_{car} = 0$$

So ii) and iii) are actually

$$\text{ii)} \quad \left(\vec{F}-\vec{G}\right)\cdot\vec{t}_{car} = \vec{F}_{car}\cdot\vec{t}_{car} - \left|\vec{F}_{dr}\right| - \left|\vec{f}_{rolling}\right|$$

$$\text{iii)} \quad \left(\vec{F}-\vec{G}\right)\cdot\vec{n}_{car} = \left|\vec{f}\right|$$

As a result, the constraints are

$$\left(\vec{F}-\vec{G}-\vec{F}_L\right)\cdot\vec{n} \geq 0$$

$$\left|k_g\right| \leq k_{\max}$$

$$v \leq v_{\max}$$

$$\left|f\right| \leq \mu\left|\vec{F}_N\right|$$

$$ma_{\min} \leq \vec{F}_{car}\cdot\vec{t}_{car} \leq ma_{\max}$$

where k_g is the geodesic curvature. According to the definition of curvature, $k = \left|\frac{d^2\vec{x}_{car}}{ds^2}\right|$. We define here

$$k_g = \frac{d^2\vec{x}_{car}}{ds^2}\cdot\vec{n}_{car}\,.$$

Similar to 2-D situation, the curvature constraint is used to avoid physically infeasible turning. The geodesic curvature is explained in Figure 2.8 below.

The reason for using the geodesic curvature is that we are looking at a 3-D road surface in a way which is similar to what we have done for a 2-D road. The geodesic curvature is the component of the curvature in the plane of the surface of the road.

For example, if a car is moving along the equator of a sphere with radius r, then its geodesic curvature is always 0, although in 3-D space it has a curvature equal to $\frac{1}{r}$.

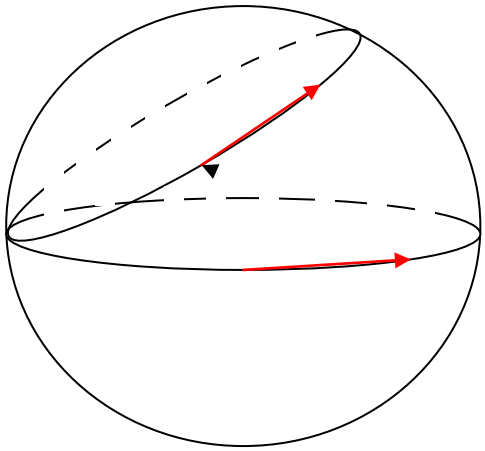


Figure 2.8. Explanation for geodesic curvature using a sphere. The geodesic curvature is zero for a path of minimum length. The marked path has some curvature in the plane tangential to the surface.

The total acceleration $\vec{a}$ can be decomposed into two components:

$$\begin{aligned}\vec{a} &= \frac{d^2\vec{x}_{car}}{dt^2} \\ &= v \cdot \frac{d}{ds}\left(v\frac{d\vec{x}_{car}}{s} \right) \\ &= v\frac{dv}{ds}\frac{d\vec{x}_{car}}{ds} + v^2\frac{d^2\vec{x}_{car}}{ds^2}\end{aligned}$$

$v\frac{dv}{ds}\frac{d\vec{x}_{car}}{ds}$ is the tangential component of $\vec{a}$, and $v^2\frac{d^2\vec{x}_{car}}{ds^2}$ is the orthogonal (to the motion of the car) component of $\vec{a}$.

Let's look at the relation (i).

$$\vec{F}\cdot\vec{n} = m\vec{a}\cdot\vec{n} = mv^2\frac{d^2\vec{x}_{car}}{ds^2}\cdot\vec{n}$$

$$\vec{G}\cdot\vec{n} = m\cdot g\cdot n_z,$$

where n_z is the z-component of $\vec{n}$.

As $\frac{d^2\vec{x}_{car}}{ds^2}\cdot\vec{n}$ is not a convenient form, we will do some reformulation to it, as follows.

$$\frac{d\vec{x}_{car}}{ds}\cdot\vec{n} = 0$$

Taking the derivative with respect to s on both sides of the equation, we have

$$\frac{d^2\vec{x}_{car}}{ds^2}\cdot\vec{n}+\frac{d\vec{x}_{car}}{ds}\cdot\frac{d\vec{n}}{ds}=0$$

$$\frac{d^2\vec{x}_{car}}{ds^2}\cdot\vec{n}=-\frac{d\vec{x}_{car}}{ds}\cdot\frac{d\vec{n}}{ds}.$$

Substitute this into $\vec{F}\cdot\vec{n}=m\vec{a}\cdot\vec{n}=mv^2\frac{d^2\vec{x}_{car}}{ds^2}\cdot\vec{n}$ and we get

$$\vec{F}\cdot\vec{n}=-mv^2\frac{d\vec{x}_{car}}{ds}\cdot\frac{d\vec{n}}{ds}$$

$$\left(\vec{F}-\vec{G}-\vec{F}_L\right)\cdot\vec{n}=-mv^2\frac{d\vec{x}_{car}}{ds}\cdot\frac{d\vec{n}}{ds}+mgn_z=\left|F_N\right|\geq 0$$

Here we require that the car will land on the track. “Flying” off the track is not allowed.

$$\vec{f}\cdot\vec{n}_{car}\leq\mu\left(-mv^2\frac{d\vec{x}_{car}}{ds}\cdot\frac{d\vec{n}}{ds}+mgn_z+L\right)$$

$$\Rightarrow\frac{1}{m}\vec{f}\cdot\vec{n}_{car}\leq\mu\left(-v^2\frac{d\vec{x}_{car}}{ds}\cdot\frac{d\vec{n}}{ds}+gn_z+L\right)$$

where L is the constant such that the acceleration from down force is

$$a_L=Lv^2$$

The friction force also satisfies

$$\frac{1}{m}\vec{f}\cdot\vec{n}_{car}=k_g v^2+g\cdot n_{car_z}$$

As a result,

$$k_g v^2+g\cdot n_{car_z}\leq\mu\left(-v^2\frac{d\vec{x}_{car}}{ds}\cdot\frac{d\vec{n}}{ds}+gn_z+L\right)$$

$$k_g v^2+g\cdot n_{car_z}\geq-\mu\left(-v^2\frac{d\vec{x}_{car}}{ds}\cdot\frac{d\vec{n}}{ds}+gn_z+L\right)$$

Reformulating the inequality constraints, we have

$$\left(k_g+\mu\frac{d\vec{x}_{car}}{ds}\cdot\frac{d\vec{n}}{ds}-\mu L\right)v^2\leq\mu gn_z-g\cdot n_{car_z}$$

$$\left(k_g-\mu\frac{d\vec{x}_{car}}{ds}\cdot\frac{d\vec{n}}{ds}-\mu L\right)v^2\geq-\mu gn_z-g\cdot n_{car_z}$$

The constraints for the car engine in the tangential direction of the car's movement are as follows:

$$v\frac{dv}{ds} + gt_{car_z} = a_{car} - Dv^2 - \mu_r\left(gn_z + \left(L - \vec{t}_{car}\cdot\frac{d\vec{n}}{ds}\right)v^2\right)$$

$$\Rightarrow a_{\min} \le v\frac{dv}{ds} + gt_{car_z} + Dv^2 + \mu_r\left(gn_z + \left(L - \vec{t}_{car}\cdot\frac{d\vec{n}}{ds}\right)v^2\right) \le a_{\max}$$

where D is the constant such that the acceleration from drag force is

$$a_D = Dv^2$$

Thus we obtain the constraints for velocity on a 3-dimensional track.

2.3 Representation of the racing tracks and racing lines

An important question is how to represent the racing tracks. We will discuss the ways to represent both 2-D tracks and 3-D tracks.

2.3.1 Representation of 2-D tracks

For a 2-dimensional track, we are keeping the information of center line and width. Many points along the line are used to represent the center of the racing track. In this thesis, at all parts of the track the width is the same. However, it is also convenient to keep a width $w(i)$ for every point i. The points on the center line can be stored in (x, y) format. This is illustrated in Figure 2.9.

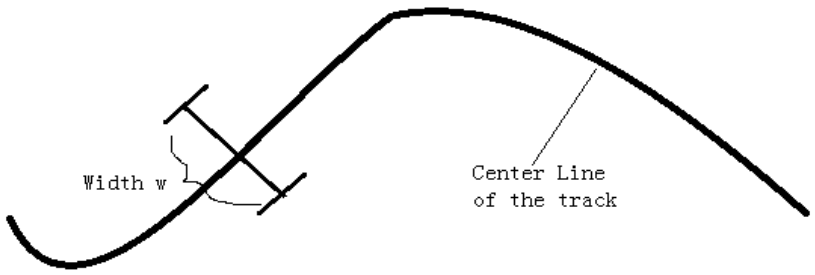


Figure 2.9. Using center line and width to describe a 2-D racing track. The outside line and inside line information can be calculated from the center line and the width.

We use s, the distance travelled at some point along the center line (initially 0), to locate the point $\big(x(s), y(s)\big)$.

2.3.2 Representation of 3-D tracks

3-dimensonal tracks are represented in a way similar to 2-dimensional tracks. Let's take a look at several examples of real-life racing tracks.

NASCAR is a very popular racing game in North America and the cars are more ordinary in appearance. From the outside, NASCAR racing cars look very similar to normal sports cars on the street, but from the inside they are completely different. NASCAR tracks are usually very simple and do not contain many turns.

Figure 2.10 shows the track and seats of Atlanta motor speedway, and Figure 2.11 is a real photo of the track. It only has two very large corners and the track is very wide at all places.

Figure 2.10. NASCAR Atlanta Motor Speedway
(Image source: http://www.nascar.com/races/tracks/ams/)

Figure 2.11. NASCAR Atlanta Motor Speedway photo
(Image source: http://sheilalovesnascar.files.wordpress.com/2009/09/atlanta-6651.jpg)

In contrast, the F1 racing tracks are not so simply shaped. They focus more on high requirements for fast cornering skills. Many F1 racing tracks are distributed in a larger area of a city, and make more use of the existing streets. Figure 2.12 is one of the most complicated F1

racing tracks, called Circuit de Catalunya in Barcelona, Spain. Figure 2.13 is the projection of the 3-D track to the 2-D earth plane.

Figure 2.12. F1 Catalunya racing track (Image source:http://www.loxlee-loves-engines.com/blog /wp-content/uploads/2009/11/circuit-de-catalunya-barcelona.jpg)

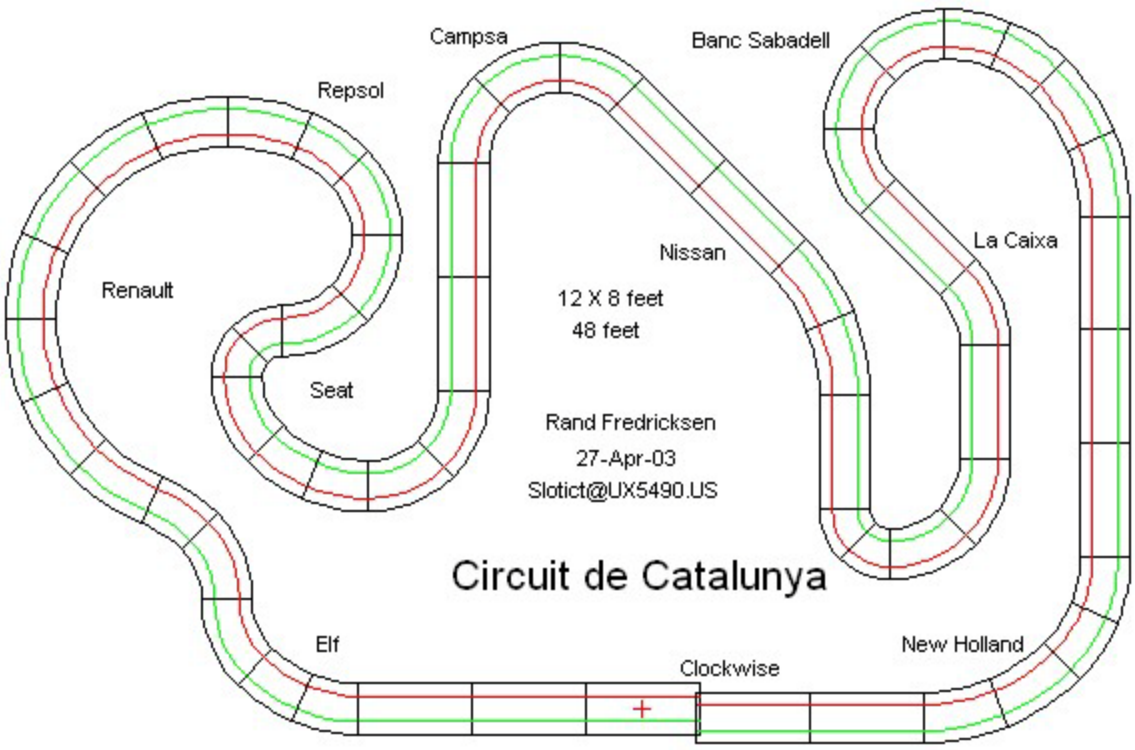


Figure 2.13. 2-D projection of the Catalunya track (Image source: http://www.ddavid.com/slot-car-gallery/images/tracks/005_circuit_de_catalunya_12x8_48.gif)

Another example is the F1 racing track in Monaco, which is unique in that all the tracks are also normal city streets (Figure 2.14). The racing track is thus in an interesting shape, too. It has a very sharp turn, almost 180 degrees, at the east end of the track (as noted by the orange circle).

Figure 2.14. F1 Monaco racing track (Graph source: http://www.loxlee-loves-engines.com/blog/wp-content/uploads/2009/11/circuit-de-catalunya-barcelona.jpg)

From the racing tracks sourced above, here are several useful conclusions:

1) In NASCAR racing, the tracks have very simple geometry. In F1, the tracks are more complicated and emphasize corners.
2) In NASCAR racing, the 3D aspects of tracks are mostly in the leaning at corners. In F1 racing, the 3D aspects of tracks are mostly in the small up and down slopes along the way. The tracks are more horizontal.
3) The length of a lap is usually less than or equal to 5 miles. The tracks are wide at all places.

An effective way of representing three-dimensional racing tracks is to use an array of discrete points to keep the center line of the track, and use the leaning angles alpha at each point to represent the leaning situation of the track (Figure 2.15).

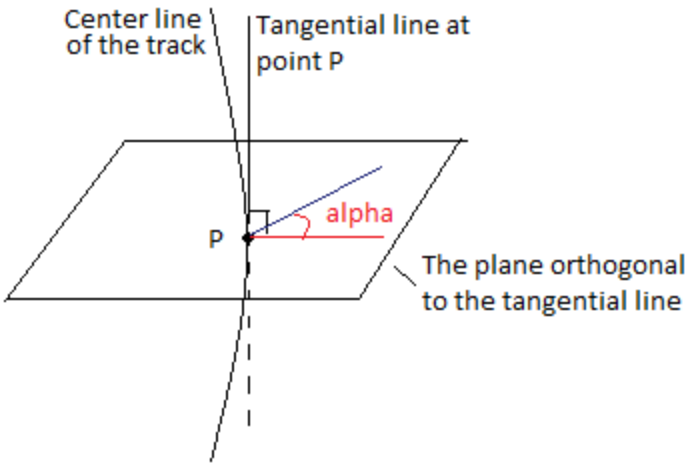


Figure 2.15. Representation of racing track by center line and angle alpha. Alpha is the angle to the horizontal plane.

2.3.3 Test cases of different racing tracks

We are going to test using the following four shapes of curves.

1) Two semi-circles connected by two straight lines (Figure 2.16).

Figure 2.16. Soccer field racing track

2) A more complex combination of parts of circles and lines (Figure 2.17).

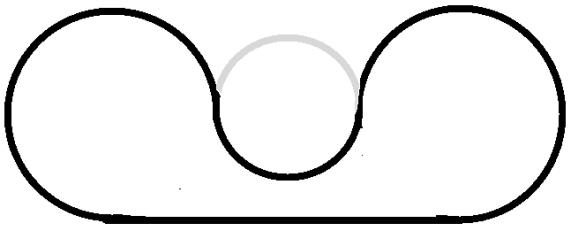


Figure 2.17. Flower shape racing track

3) The ellipse racing track (Figure 2.18).

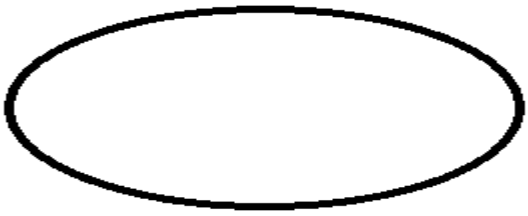


Figure 2.18. The ellipse racing track

4) Rounded square track.

The rounded square track is a rectangle with edge lengths (m+2r) and (n+2r) and rounded corners of radius r (Figure 2.19). The MATLAB plot of the center line of a 2-D rounded square track is shown in Figure 2.20, and the MATLAB plot of the center line of a 3-D rounded square track is shown in Figure 2.21.

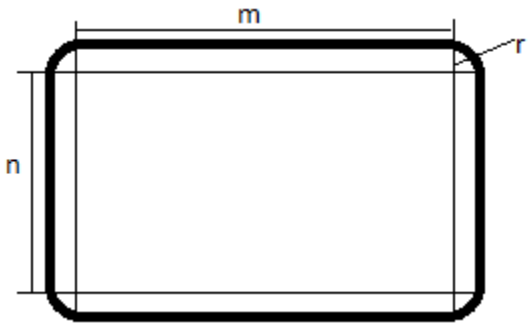


Figure 2.19. The rounded square racing track

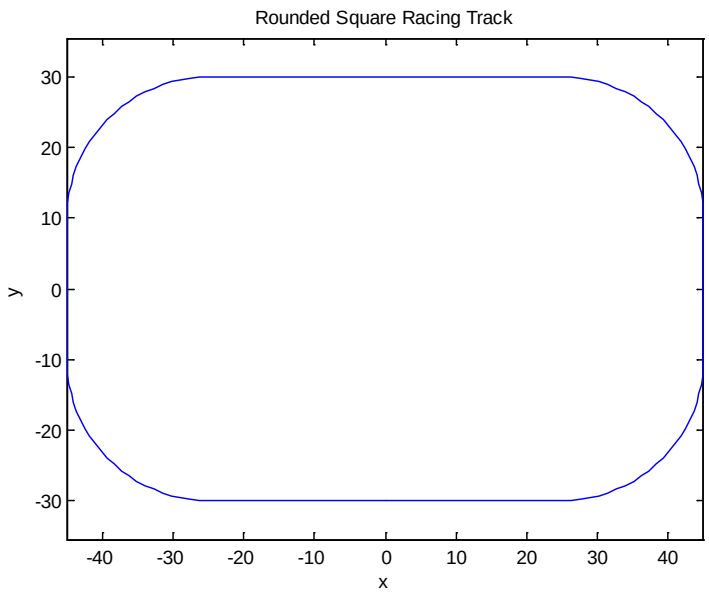


Figure 2.20. Plot of the center line of a 2-D rounded square racing track in Matlab.

Then we add some slope to the straight part so that it is not totally flat. We will later compare the two situations.

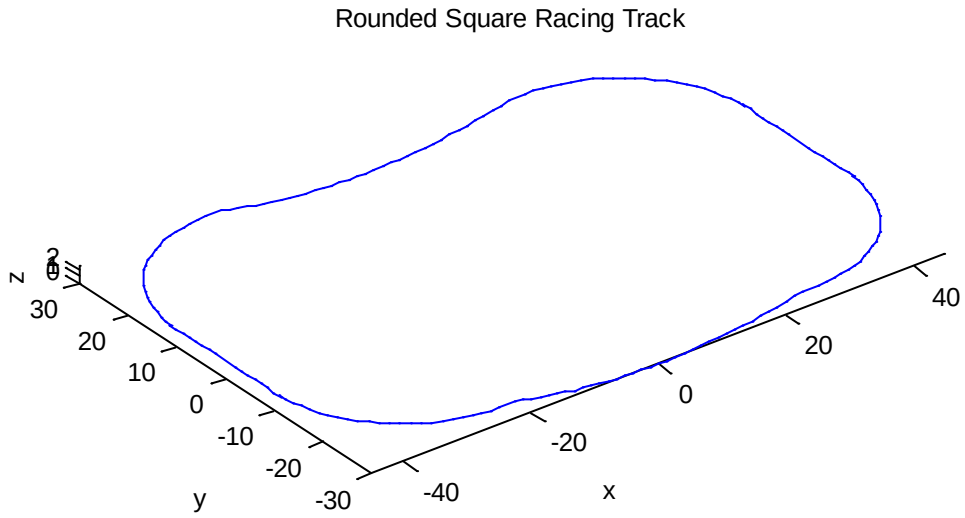


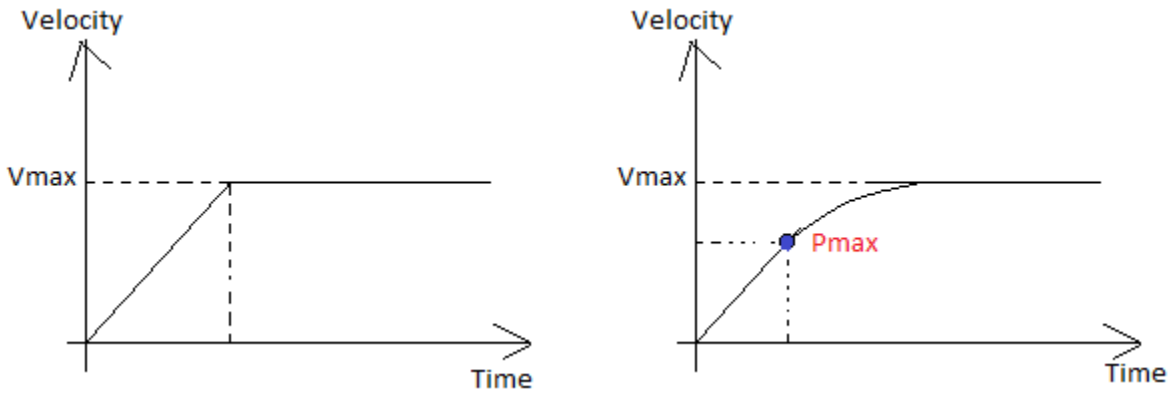
Figure 2.21. Plot of the center line of a 3-D rounded square racing track in MATLAB.

2.4 Problem modification using power constraint

In Chapter 2.1 and 2.2, we introduced the 2-dimensional and 3-dimensional problem formulations. Some results will be shown in Chapter 7. The results achieved using the original problem formulation are not perfect. There is one constraint that has not been considered yet - power.

Under maximum and minimum acceleration constraints, the speed can go from v(i) to v(i+1) from point i to point i+1. However, it may be hard for drivers to really implement a given set of velocities when they are driving. We can make this a little easier to implement if the upper velocity constraint is not for acceleration, but for power.

From $P = F \cdot v = m \cdot a \cdot v$ we can see that when the velocity is large, the pull force of the car is constrained by v, and so is the acceleration. The velocity will move in a smoother way because with large velocity, acceleration cannot be very high, and the velocity can only change relatively slowly. The comparison of the two kinds of constraints for a car speeding up from zero velocity is in Figure 2.22



(a) (b)

Figure 2.22. Comparison of v-t graph using acceleration constraint and power constraint

The comparison of the two figures above shows the difference between using maximum acceleration as a constraint and using maximum power.. In the situation where the car speeds up from zero to v_{max}, in the left figure the acceleration is constantly a_{max} and the velocity curve is not differentiable at the turning point where v_{max} is achieved. However, in the right figure,

the acceleration is also changing continuously and the velocity curve is differentiable at every point.

The problem formulation will need a slight change according to the change of constraints. For example, in the 2-D situation, the problem changes from

$$\begin{aligned} &\text{Minimize } \int \frac{1}{v} ds \\ &s.t. \qquad kv^2 - \mu g \le 0 \\ &\int k ds = \Delta\theta \\ &k \le k_{\max} \\ &v \le v_{\max} \\ &a_{\min} \le a \le a_{\max} \end{aligned}$$

to

$$\begin{aligned} &\text{Minimize } \int \frac{1}{v} ds \\ &s.t. \qquad kv^2 - \mu g \le 0 \\ &\int k ds = \Delta\theta \\ &k \le k_{\max} \\ &v \le v_{\max} \\ &Fv \le P_{\max} \\ &F = ma \\ &a \ge a_{\min} \end{aligned}$$

Notice that here a is not the total acceleration in the tangent direction, but the component provided by the car engine.

The 2006 F1 cars have a power-to-weight ratio of 0.93 kW/kg. However the massive power cannot be converted to motion at low speeds due to traction loss, and the usual figure is 2 seconds to reach 100 km/h. After about 130 km/h, traction loss is minimal due to the combined effect of the car moving faster and the down force; hence the car continues accelerating at a very high rate. [2]

3. Optimal cornering with the Euler spiral

In this chapter we will discuss the Euler spiral method and its implementation and results for the optimal racing line problem.

3.1 Euler spiral method

An Euler spiral is a curve whose curvature changes linearly with its curve length. Euler spirals are widely used for connecting the geometry between a tangent and a circular curve. The principle of this transition is: the start point will follow a segment of the Euler spiral to reach the end point, and their tangential direction vectors are angle $\Delta\theta$ apart, where $\Delta\theta$ is the desired turning angle.

The Euler spiral comes close to the minimization of $E = \int k^2 ds$, but does not actually achieve the minimum value.[5] An example of a double end Euler spiral is in Figure 3.1.

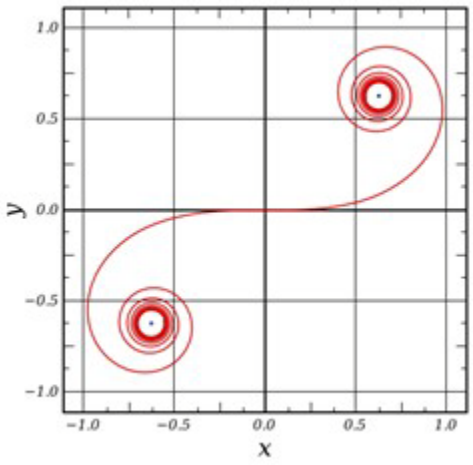


Figure 3.1. A double-end Euler spiral[6]

A small segment of the double-end Euler spiral above is taken to approximate the racing line.

Some symbols that are used in describing the Euler spiral are as follows:

r - Radius at a certain point (reciprocal of the curvature)

r_c - Radius of the curve at the end of the spiral

θ - Angle of curve from beginning of spiral (infinite r) to a particular point on the spiral

L - Length measured along the spiral curve from its initial position

L_s - Length of the spiral curve

For an Euler spiral,

$$k = \frac{1}{r}$$

$$r = \frac{dL}{d\theta}$$

$$\frac{1}{r} = \frac{d\theta}{dL} \propto L$$

$$\therefore rL = r_c L_s = const$$

Let $a = 1/\sqrt{2r_c L_s}$, then

$$\frac{d\theta}{dL} = 2a^2 L$$

Letting $s = aL$, x(s) and y(s) can be written in the Fresnel integral form

$$\begin{cases} x(s) = \frac{1}{a}\int_0^s \cos s^2 ds \\ y(s) = \frac{1}{a}\int_0^s \sin s^2 ds \end{cases}$$

The Taylor expansions of these functions are as follows:

$$x(s) = \frac{1}{a}(s - \frac{s^5}{5\times 2!} + \frac{s^9}{9\times 4!} - \frac{s^{13}}{13\times 6!} + \cdots)$$
$$y(s) = \frac{1}{a}(s^3 - \frac{s^7}{7\times 3!} + \frac{s^{11}}{11\times 5!} - \frac{s^{15}}{15\times 7!} + \cdots)$$

The next step is to determine the speed at each point of the track. As shown in Figure 1.4, the relation between v and k is specified; v is defined as a function of k. When curvature k becomes larger (radius r becomes smaller), the maximum speed decreases.

Figure 3.2 shows a segment of the Euler spiral.

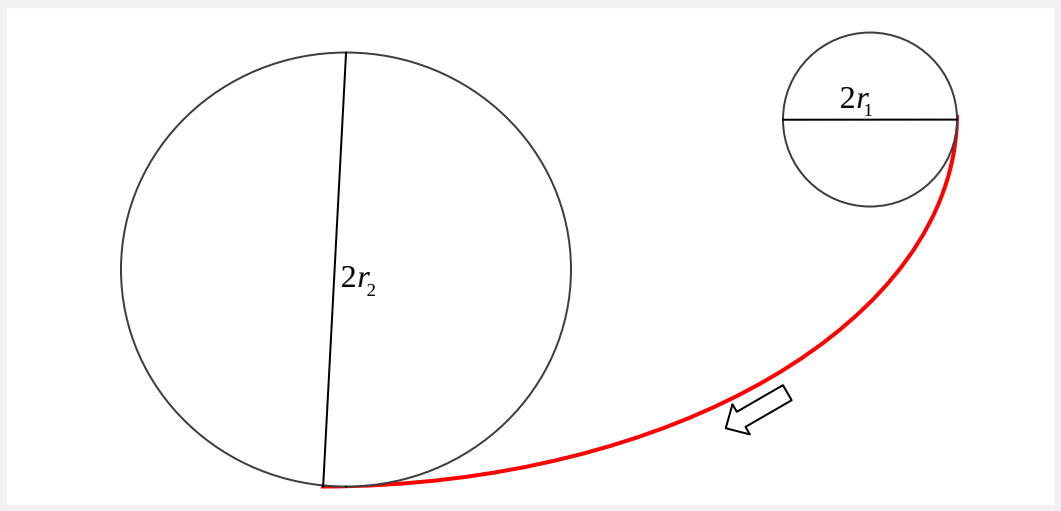


Figure 3.2. A segment of Euler spiral. The curvature is decreasing linearly along the line. At the starting point, the radius is r_1, while at the end point, the radius is r_2.

Suppose that there is a 90 degree turn. The total distance that the car travels will depend on the value of the turning radius at each point:

$$s_1 = \int ds = \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} r(\theta) d\theta$$

Curvature is changing linearly for an Euler spiral:

$$k(s) = r_1 + \frac{r_2 - r_1}{s_1} s$$

According to the expression of time $t = \int \frac{1}{v} ds$, we have

$$t = \int \frac{1}{v} ds = \int \sqrt{\frac{k(s)}{\mu g}} ds = \int \sqrt{\frac{r_1 + \frac{r_2 - r_1}{s_1} s}{\mu g}} ds$$

There is more than one Euler spiral segment connecting the start point and the end point; and according to Henry A. Watts[19] the optimal racing line always makes a smooth curve and nicely hit the inside edge of the track at some point of the turn. Thus in implementing the Euler spiral, we seek to achieve an apex.

Figure 3.3 shows an example of late apex racing line with late turn in point.

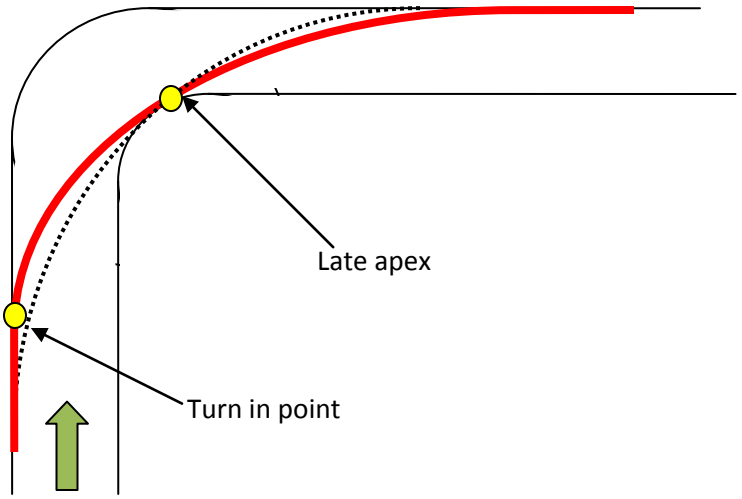


Figure 3.3. A late-apex racing line is shown by the red line. The two yellow round points are the turn in point and the late apex, respectively. To achieve a late apex there should be a late turn in point. The dot line is the center-apex racing line.

Late-apex is usually preferred. Carrying the highest average speed around corners may not actually be the quickest way around a track. If the corner leads onto a straight it can be better to take a late apex, straighten the car out early and get the power on for a high-speed exit. This is generally regarded as the best strategy for racing, with a slightly lower entry speed but a faster exit speed. The amount of grip available is the factor which determines how late you can brake and touch an apex.

3.2 Implementation results

The main idea of this approach is to use the Euler spiral to do cornering. In the experiments, 90 degree turning is tested. An interesting thing is that in this case, we are always using a fixed part of any Euler spiral, and s can be used as a deciding factor of the spiral's shape. Thus, the most important thing is to figure out the length of the part of Euler spiral that we need, shown as the variable s.

In the program, by setting an initial value for s and then changing s accordingly using bisection method, we can find an Euler spiral that is just touching the inner track boundary. Figure 3.3 shows a track with rectangular boundaries.

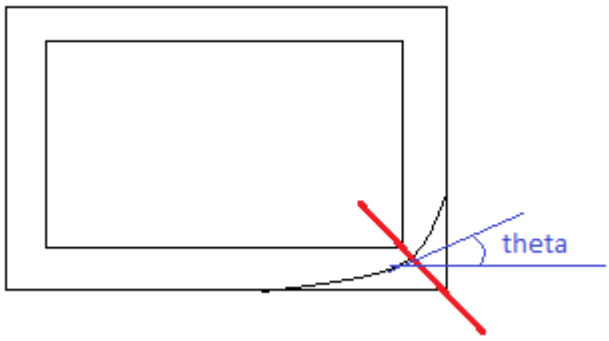


Figure 3.3. Rectangle shaped racing track. At one point the direction of the car is at an angle theta with the horizontal line

In reality, many tracks are rounded square shape as in the following images (Figure 3.4):

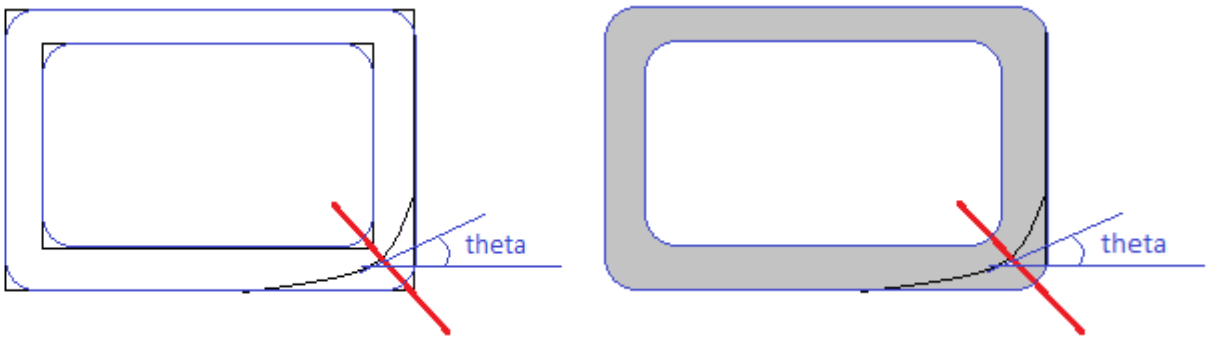


Figure 3.4. Square shape rounded corner racing track

Zooming in at one right angle for the rounded track is Figure 3.5:

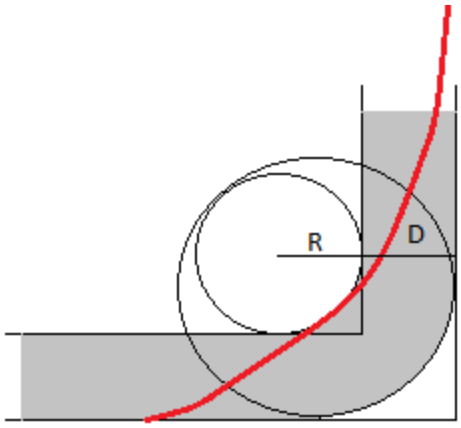


Figure 3.5. Geometry of the rounded square track corner

The following flow chart shows how the Euler spiral results are achieved (Figure 3.6).

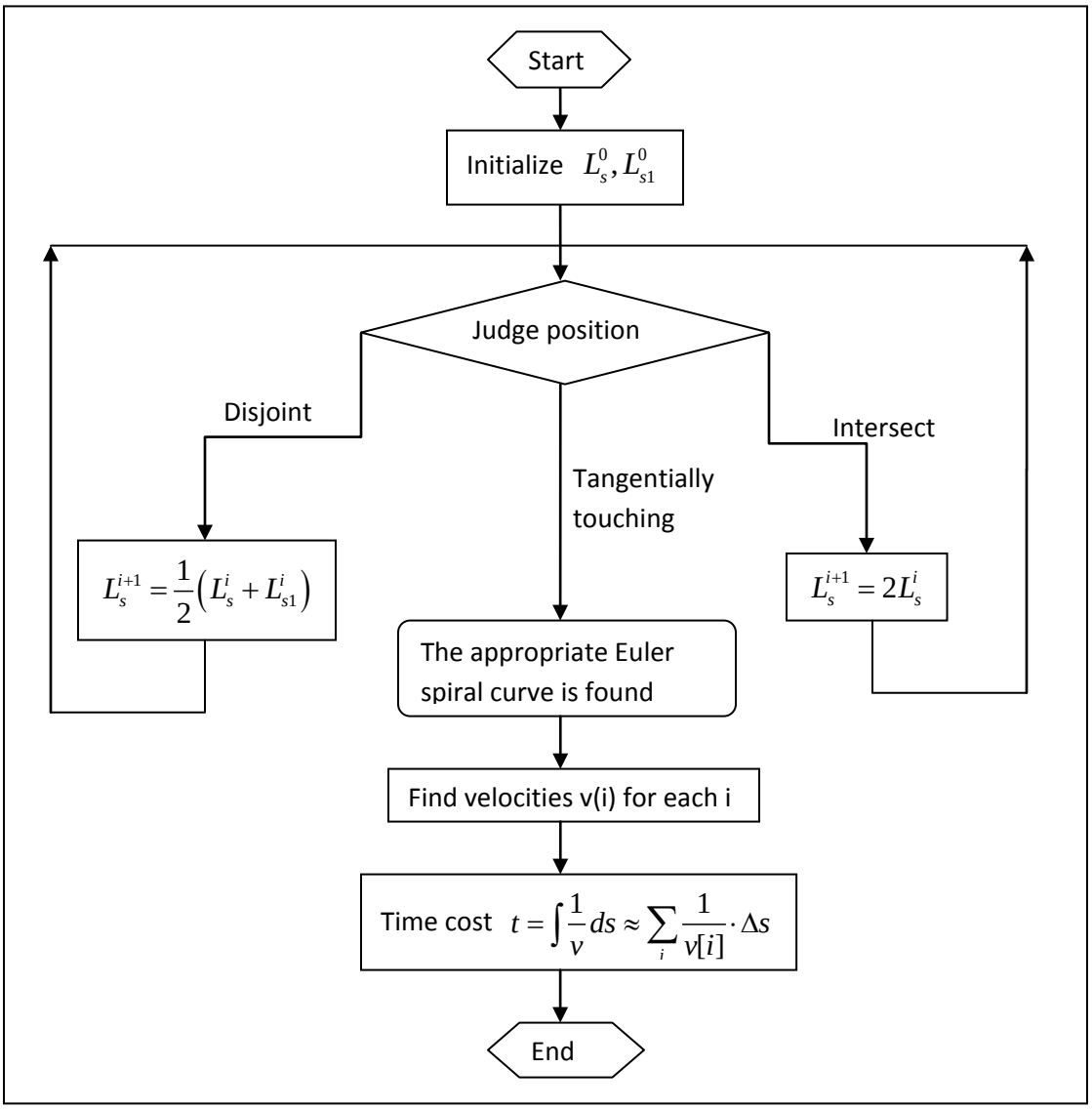
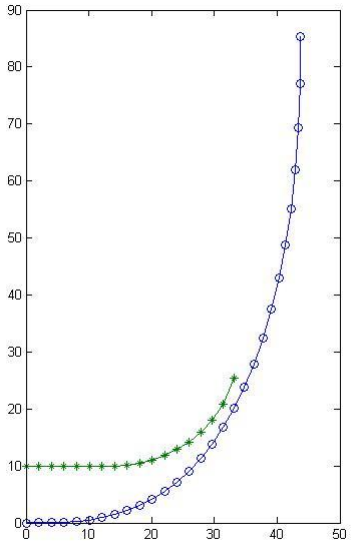
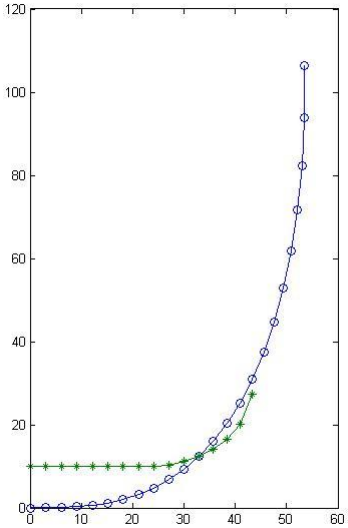
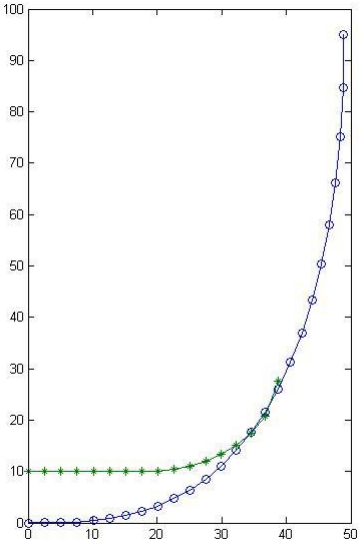
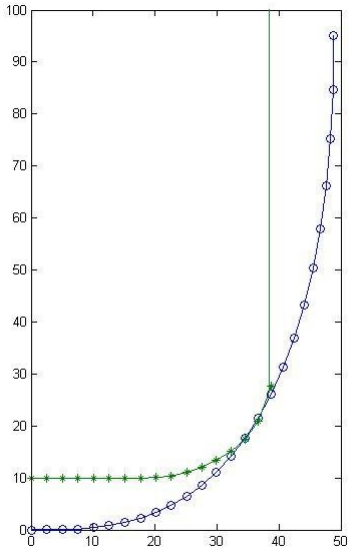


Figure 3.6. Euler spiral method flow chart

Some of the graphs generated during the process are in Table 3.1.

Table 3.1. Results of the Euler spiral cornering

Not touched yet	Touches too much
	
Correctly touches the (late) apex	**Adding the track shape**
	

The velocity graph is in Figure 3.7. We can see that the speed was originally very large, then through the corner it decreases, and then increases again. At the exit, it already achieves the maximum speed (assume the maximum speed is 70 m/s).

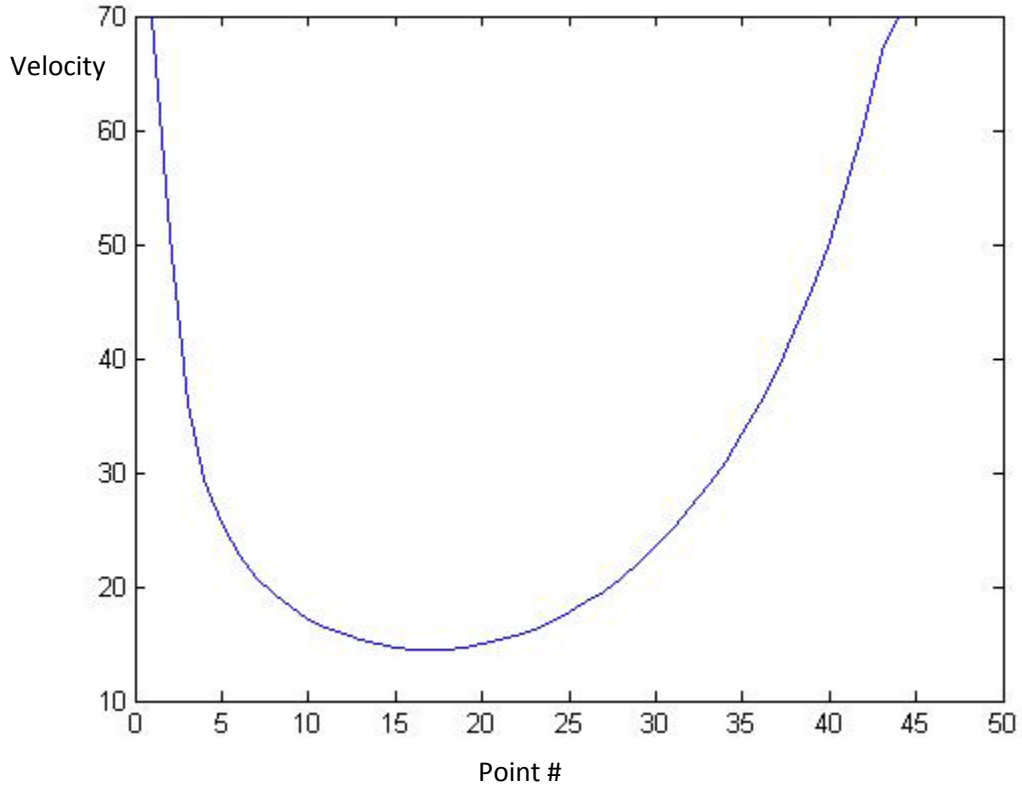


Figure 3.7. Velocity graph for Euler spiral cornering

The Euler spiral approach can be used for any angle of cornering, and the above is just an example of a 90 degree turn. A general usage of the Euler spiral method is shown in Figure 3.8. When the turning angle is θ we can turn it using a part of the Euler spiral as shown below. Sometimes the curve may not be applicable; it may go outside the track.

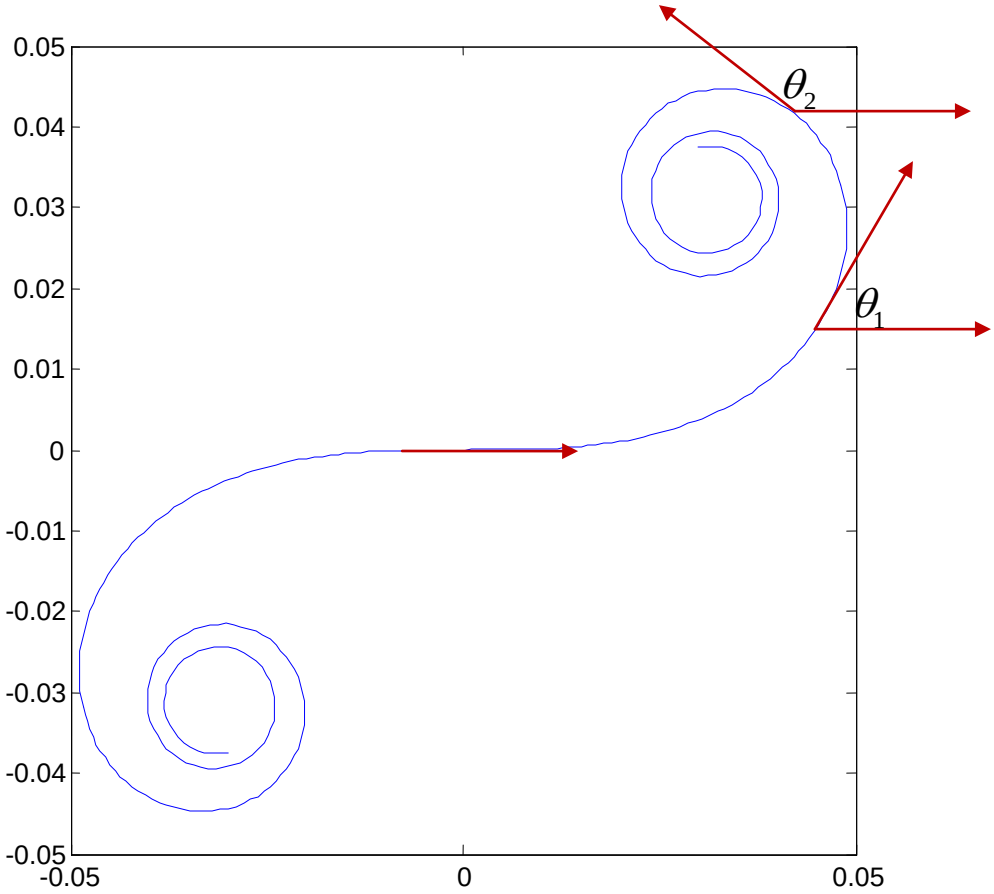


Figure 3.8. Euler spiral method for arbitrary angle cornering

4. Nonlinear programming solver approach

The problem can be formulated as a dynamic programming problem, and thus special nonlinear programming languages can be used as tools to solve the problem.

4.1 Problem formulation for nonlinear solver approach

As before, the total time cost is:

$$t = \int \frac{1}{v} ds$$

Assume the ideal vacuum 2-D track condition:

$$v^2 k \le \mu g$$

$$\therefore \frac{1}{v} \ge \sqrt{\frac{k}{\mu g}}$$

$$\therefore t \ge \frac{1}{\sqrt{\mu g}} \cdot \int \sqrt{k} ds$$

The optimal racing line of shortest time cost is very similar to the racing line of the smallest $E = \int \sqrt{k} ds$. In practice the problem can be simplified to minimizing $E = \int \sqrt{k} ds$. Of course, there are some differences between the time-best racing line and the $\int \sqrt{k} ds$ -best racing line, so the result will not be precise. This method can run very fast for large-scale problems. If there are many points along the center line, the large-scale optimization problem can be easily solved by nonlinear solvers.

Consider the nonlinear programming problem:

(1) Objective function

$$E = \sum_{i=2}^{n-1} \sqrt{k(i)} \frac{\sqrt{\left(x(i)-x(i-1)\right)^2 + \left(y(i)-y(i-1)\right)^2} + \sqrt{\left(x(i+1)-x(i)\right)^2 + \left(y(i+1)-y(i)\right)^2}}{2}$$

(2) Input parameters:

$xc(i)$ and $yc(i)$ $\quad i = 1, 2, ..., n$: The center line coordinates.

$width$: The width of the track

$k_{\max}$: Maximum curvature allowed

(3) Unknown variables

$x(i)$ and $y(i)$ $\quad i = 1, 2, ..., n$

$k(i)$ $\quad i = 1, 2, ..., n$

(4) Constraints:

$$k(i) = \frac{\left| \frac{y(i) - y(i-1)}{x(i) - x(i-1)} \right|}{\left[1 + \left(\frac{y(i-1) + y(i+1) - 2y(i-1)}{x(i) - x(i-1)} \right)^2 \right]^{\frac{3}{2}}} \qquad i = 2, 3, \ldots (n-1)$$

$$k(i) \le k_{\max} \qquad i = 2, 3, \ldots (n-1)$$

$$\sqrt{\left(x(i) - xc(i) \right)^2 + \left(y(i) - yc(i) \right)^2} \le \frac{1}{2} width \qquad i = 1, 2, \ldots, n$$

$$\dot{xc}(i)\left(x(i) - xc(i) \right) + \dot{yc}(i)\left(y(i) - yc(i) \right) = 0 \qquad i = 1, 2, \ldots, n$$

There is a problem that $x(i) - x(i-1)$ might be zero. To avoid this, we can use another way to represent the curvature variable k, as follows:

$$k(s)=\frac{1}{r}=\frac{d\theta}{ds}=\frac{d\arctan\left(\frac{dy}{dx}\right)}{ds}$$

$$=\frac{1}{1+\left(\frac{dy}{dx}\right)^2}\cdot\frac{d\left(\frac{dy}{dx}\right)}{ds}=\frac{1}{1+\left(\frac{dy}{dx}\right)^2}\cdot\frac{d\left(\frac{\frac{dy}{ds}}{\frac{dx}{ds}}\right)}{ds}$$

$$=\frac{1}{1+\left(\frac{\dot{y}}{\dot{x}}\right)^2}\cdot\frac{\ddot{y}\dot{x}-\ddot{x}\dot{y}}{\dot{x}^2}=\frac{\ddot{y}\dot{x}-\ddot{x}\dot{y}}{\dot{x}^2+\dot{y}^2}$$

$\because \quad (dx)^2+(dy)^2=(ds)^2$

$\therefore \quad \dot{x}^2+\dot{y}^2$

$\therefore \quad k(s)=\ddot{y}\dot{x}-\ddot{x}\dot{y}$

So the expression for k(i) can also be written as

$$k(i)=\frac{dx}{ds}\frac{d^2y}{ds^2}-\frac{dy}{ds}\frac{d^2x}{ds^2}$$

$$=\frac{1}{2}\left(\frac{x(i)-x(i-1)}{\sqrt{(x(i)-x(i-1))^2+(y(i)-y(i-1))^2}}+\frac{x(i+1)-x(i)}{\sqrt{(x(i+1)-x(i))^2+(y(i+1)-y(i))^2}}\right)$$

$$\cdot\frac{\frac{y(i+1)-y(i)}{\sqrt{(x(i+1)-x(i))^2+(y(i+1)-y(i))^2}}-\frac{y(i)-y(i-1)}{\sqrt{(x(i)-x(i-1))^2+(y(i)-y(i-1))^2}}}{\frac{\sqrt{(x(i+1)-x(i))^2+(y(i+1)-y(i))^2}+\sqrt{(x(i)-x(i-1))^2+(y(i)-y(i-1))^2}}{2}}$$

$$-\frac{1}{2}\left(\frac{y(i)-y(i-1)}{\sqrt{(x(i)-x(i-1))^2+(y(i)-y(i-1))^2}}+\frac{y(i+1)-y(i)}{\sqrt{(x(i+1)-x(i))^2+(y(i+1)-y(i))^2}}\right)$$

$$\cdot\frac{\frac{x(i+1)-x(i)}{\sqrt{(x(i+1)-x(i))^2+(y(i+1)-y(i))^2}}-\frac{x(i)-x(i-1)}{\sqrt{(x(i)-x(i-1))^2+(y(i)-y(i-1))^2}}}{\frac{\sqrt{(x(i+1)-x(i))^2+(y(i+1)-y(i))^2}+\sqrt{(x(i)-x(i-1))^2+(y(i)-y(i-1))^2}}{2}}$$

Simplifying the equation above, we obtain

$$k(i)=\left(\frac{x(i)-x(i-1)}{\sqrt{\left(x(i)-x(i-1)\right)^2+\left(y(i)-y(i-1)\right)^2}}+\frac{x(i+1)-x(i)}{\sqrt{\left(x(i+1)-x(i)\right)^2+\left(y(i+1)-y(i)\right)^2}}\right)$$
$$\cdot\frac{\frac{y(i+1)-y(i)}{\sqrt{\left(x(i+1)-x(i)\right)^2+\left(y(i+1)-y(i)\right)^2}}-\frac{y(i)-y(i-1)}{\sqrt{\left(x(i)-x(i-1)\right)^2+\left(y(i)-y(i-1)\right)^2}}}{\sqrt{\left(x(i+1)-x(i)\right)^2+\left(y(i+1)-y(i)\right)^2}+\sqrt{\left(x(i)-x(i-1)\right)^2+\left(y(i)-y(i-1)\right)^2}}$$

$$-\left(\frac{y(i)-y(i-1)}{\sqrt{\left(x(i)-x(i-1)\right)^2+\left(y(i)-y(i-1)\right)^2}}+\frac{y(i+1)-y(i)}{\sqrt{\left(x(i+1)-x(i)\right)^2+\left(y(i+1)-y(i)\right)^2}}\right)$$
$$\cdot\frac{\frac{x(i+1)-x(i)}{\sqrt{\left(x(i+1)-x(i)\right)^2+\left(y(i+1)-y(i)\right)^2}}-\frac{x(i)-x(i-1)}{\sqrt{\left(x(i)-x(i-1)\right)^2+\left(y(i)-y(i-1)\right)^2}}}{\sqrt{\left(x(i+1)-x(i)\right)^2+\left(y(i+1)-y(i)\right)^2}+\sqrt{\left(x(i)-x(i-1)\right)^2+\left(y(i)-y(i-1)\right)^2}}$$

Problem formulation:

$$\text{minimize} \quad \sum_{i=2}^{n-1} \sqrt{k(i)} \frac{\sqrt{\left(x(i)-x(i-1)\right)^2+\left(y(i)-y(i-1)\right)^2}+\sqrt{\left(x(i+1)-x(i)\right)^2+\left(y(i+1)-y(i)\right)^2}}{2}$$

s.t.

$$k(i)=\left(\frac{x(i)-x(i-1)}{\sqrt{\left(x(i)-x(i-1)\right)^2+\left(y(i)-y(i-1)\right)^2}}+\frac{x(i+1)-x(i)}{\sqrt{\left(x(i+1)-x(i)\right)^2+\left(y(i+1)-y(i)\right)^2}}\right)$$

$$\cdot\frac{\frac{y(i+1)-y(i)}{\sqrt{\left(x(i+1)-x(i)\right)^2+\left(y(i+1)-y(i)\right)^2}}-\frac{y(i)-y(i-1)}{\sqrt{\left(x(i)-x(i-1)\right)^2+\left(y(i)-y(i-1)\right)^2}}}{\sqrt{\left(x(i+1)-x(i)\right)^2+\left(y(i+1)-y(i)\right)^2}+\sqrt{\left(x(i)-x(i-1)\right)^2+\left(y(i)-y(i-1)\right)^2}}$$

$$-\left(\frac{y(i)-y(i-1)}{\sqrt{\left(x(i)-x(i-1)\right)^2+\left(y(i)-y(i-1)\right)^2}}+\frac{y(i+1)-y(i)}{\sqrt{\left(x(i+1)-x(i)\right)^2+\left(y(i+1)-y(i)\right)^2}}\right)$$

.

$$\frac{\frac{x(i+1)-x(i)}{\sqrt{\left(x(i+1)-x(i)\right)^2+\left(y(i+1)-y(i)\right)^2}}-\frac{x(i)-x(i-1)}{\sqrt{\left(x(i)-x(i-1)\right)^2+\left(y(i)-y(i-1)\right)^2}}}{\sqrt{\left(x(i+1)-x(i)\right)^2+\left(y(i+1)-y(i)\right)^2}+\sqrt{\left(x(i)-x(i-1)\right)^2+\left(y(i)-y(i-1)\right)^2}}$$

$$i=2,3,\ldots(n-1)$$

$$k(i)\le k_{\max} \qquad i=2,3,\ldots(n-1)$$

$$\sqrt{\left(x(i)-xc(i)\right)^2+\left(y(i)-yc(i)\right)^2}\le\frac{1}{2}width \qquad i=1,2,\ldots,n$$

$$\frac{1}{2}\left(\frac{xc(i)-xc(i-1)}{\sqrt{\left(xc(i)-xc(i-1)\right)^2+\left(yc(i)-yc(i-1)\right)^2}}+\frac{xc(i+1)-xc(i)}{\sqrt{\left(xc(i+1)-xc(i)\right)^2+\left(yc(i+1)-yc(i)\right)^2}}\right)$$

$$\cdot\left(x(i)-xc(i)\right)+\frac{1}{2}\left(\frac{yc(i)-yc(i-1)}{\sqrt{\left(xc(i)-xc(i-1)\right)^2+\left(yc(i)-yc(i-1)\right)^2}}+\frac{yc(i+1)-yc(i)}{\sqrt{\left(xc(i+1)-xc(i)\right)^2+\left(yc(i+1)-yc(i)\right)^2}}\right)$$

$$\cdot\left(y(i)-yc(i)\right)=0 \qquad i=1,2,\ldots,n$$

Assume that $k_{\max}=0.5$. Then $k(i)$ should be less than or equal to 0.5 to ensure that the turning radius is larger than or equal to 2.

4.2 Solving the problem using existing commercial nonlinear solvers

The key point of this method is to make use of existing optimization toolboxes or languages. We code it in AMPL and define the input and output data. Since it is a complicated nonlinear problem, we should use nonlinear solvers. Current nonlinear solvers for AMPL do not support the square root operator in the objective function, so we avoid using the square root operators by defining all step sizes to be the same ds. Varying step sizes are not allowed with the existing AMPL solvers because they cannot handle the situation where there are variables inside the square root sign, such as $\sqrt{f(x)}$ where x is a variable.

Assume that the distance between any two neighboring points is equal to ds. For a given curve, assume that ds remains unchanged regardless of the racing line taken. Also, square root of k is changed to |k| in the objective function. In this case, our problem will be simplified a lot, as below.

$$\text{minimize} \quad \sum_{i=2}^{n-1} \left| k(i) \right| \cdot ds$$

$$s.t. \qquad k(i) = \frac{x(i+1)\left(y(i)+y(i-1)\right) + x(i)(y(i+1)-y(i-1)) - x(i-1)(y(i+1)-y(i))}{(ds)^3} \qquad i = 2,3,\ldots(n-1)$$

$$k(i) \le k_{\max} \qquad i = 2,3,\ldots(n-1)$$

$$\left(x(i)-xc(i)\right)^2 + \left(y(i)-yc(i)\right)^2 \le \frac{1}{4} width^2 \qquad i = 1,2,\ldots,n$$

$$\left(x(i)-xc(i)\right) + k(i)\left(y(i)-yc(i)\right) = 0 \qquad i = 1,2,\ldots,n$$

Different nonlinear solvers are tested here, including MINOS, IPOPT, LOQO and SNOPT. IPOPT and LOQO are both nonlinear solver using interior point methods.

IPOPT[21] (Interior Point Optimizer) is a relatively new nonlinear solver. It is used for large scale nonlinear optimization of continuous systems. It implements a primal-dual interior point method, and uses line searches based on Filter methods. It is designed to exploit first derivatives and Hessians if provided. If no Hessians are provided, IPOPT will approximate them using Quasi-Newton methods. In the experimental results, we can see that this method tends to

converge to a locally infeasible point with some curvy tracks like the Flower Track but works well with simpler tracks.

LOQO[22] and SNOPT [23] are provided in the MIT Athena cluster where we did some tests. SNOPT is very effective for a nonlinear problem whose functions and gradients are expensive to evaluate. It needs the functions to be smooth, but can deal with the problem formulation without the equal-*ds* assumption. However, with the assumption that the distances between points are all equal, it does not work especially well. LOQO is a system for solving smoothly constrained optimization problems. The problems can be linear or nonlinear, convex or non-convex. The only real restriction is that the functions defining the problem be smooth (at the points evaluated by the algorithm). If the problem is convex, LOQO finds a globally optimal solution. Otherwise, it finds a locally optimal solution near to a given starting point. The problem with this method is also that often it will converge to a local optimum.

MINOS[24] is used for solving large-scale optimization problems, both linear and nonlinear. It is especially effective for linear programs and for problems with a nonlinear objective function and sparse linear constraints. MINOS can also process large numbers of nonlinear constraints. The nonlinear functions should be smooth but need not be convex. For problems with nonlinear constraints, MINOS uses a sparse SLC algorithm, which is a projected Lagrangian method, related to Robinson's method. It solves a sequence of sub problems in which the constraints are linearized and the objective is an augmented Lagrangian (involving all nonlinear functions). Convergence is rapid near a solution.

Comparing the pros and cons of the solvers mentioned above, MINOS is the one that fits our need since our problem has an objective function for which it is not hard to calculate derivatives, but there are many non-linear constraints. It also runs faster than the other methods.

We write the module and data files, and run the program under the AMPL environment (Figure 4.1).

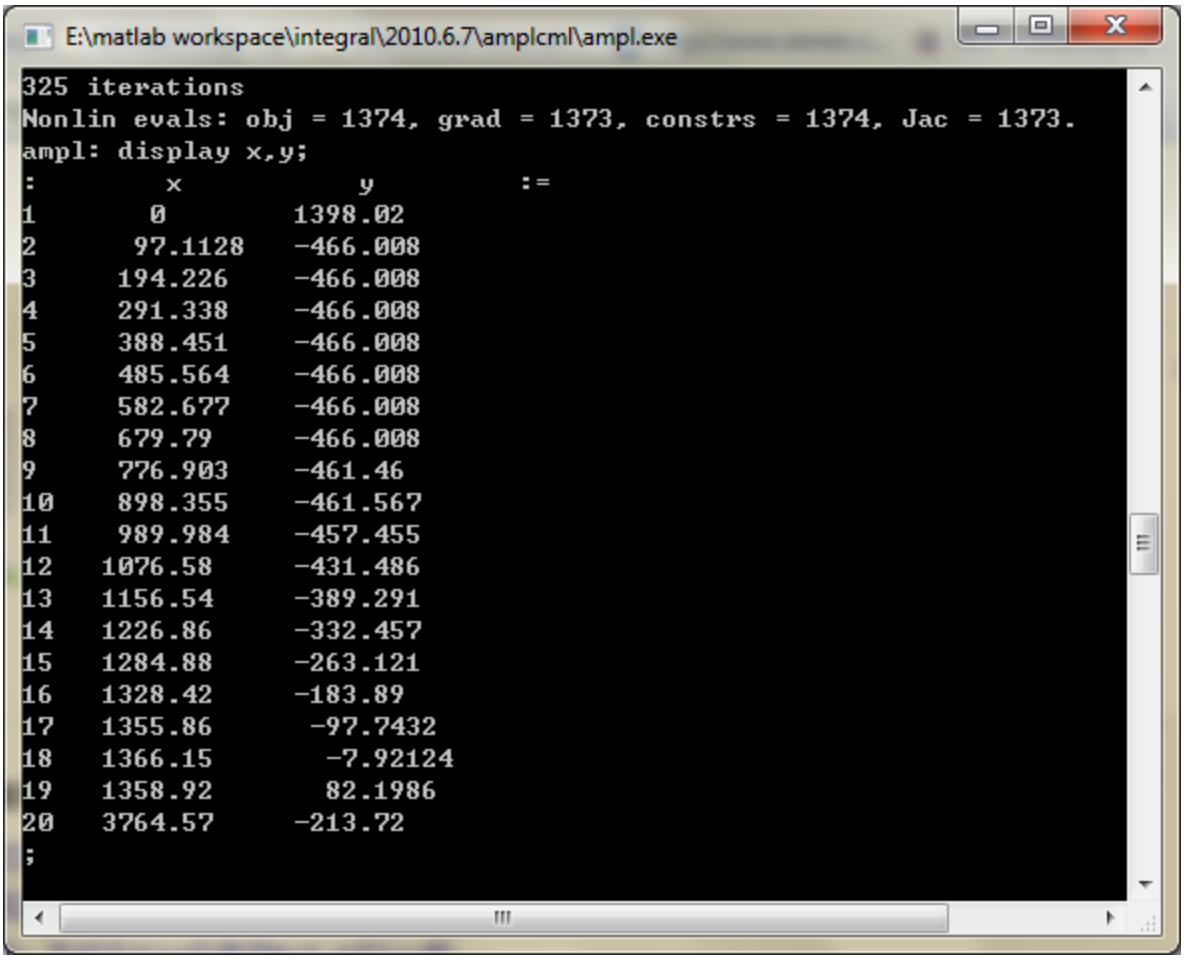


Figure 4.1. Solving the nonlinear programming problem in the AMPL environment

We use the MINOS solver, and the optimal racing line generated for a flower track is drawn in Figure 4.2. The blue line is the center line of the track, and red line is the racing line.

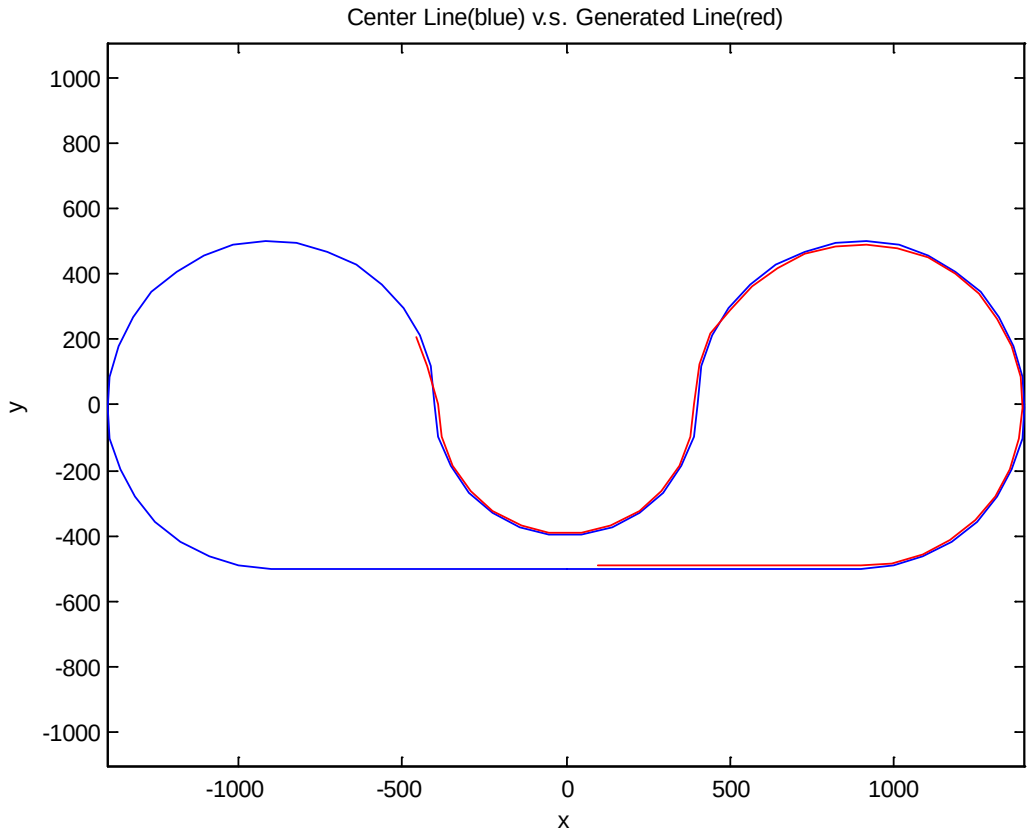


Figure 4.2. Center line with generated racing line by AMPL

The result makes sense here. Note that there are no initial values assigned to x[i], i=41,..,50, but for x[i], i=1,..40 the initial values are the center line points. The reason for the initialization of some values is to avoid being trapped at a local optimum and not being able to jump out. Driving through the center line of a track is a reasonable starting point for the optimization process.

Without initialization, sometimes the solver will stop and show the message “MINOS 5.5: the current point cannot be improved”. This shows the nonlinear programming problem is trapped at a local minimum. In order to avoid the large possibility of being trapped at a local optimum instead going for the global minimum, the following two steps are taken:

(1) Set an initial guess which is reasonably close

(2) Add penalty constraints to avoid unnecessary noise. We can use second derivatives of x and y to be penalties. Sharp changes are not preferred.

Zooming in, we can see more detailed features of the optimal result given by the solver in Figure 4.3.

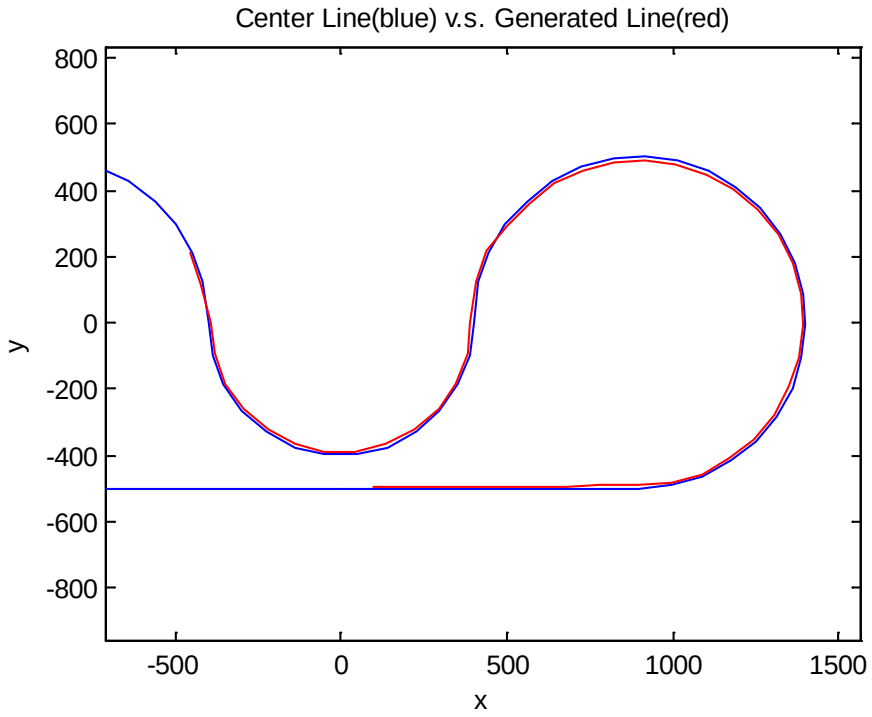


Figure 4.3. A zoom-in view of Figure 4.2

Zooming in Figure 4.2 to get Figure 4.3, we can clearly see the apex strategies that the optimal racing line takes to achieve shortest time cost. It has two apexes, one at each corner, and both are late apexes.

Here we use the AMPL on MIT Athena cluster environment so it will not have constraints on the number of variables.

Apart from the MINOS solver on AMPL, we can also use the Optimization Toolbox from MATLAB. The Optimization Toolbox provides a set of functions including nonlinear minimization, linear least squares, nonlinear least squares, nonlinear zero finding, minimization of matrix problem, etc. As our problem has a nonlinear objective with linear and nonlinear constraints, we use the FMINCON function provided by the toolbox. FMINCON finds a constrained minimum of a function of several variables. FMINCON attempts to solve problems of the form:

$$\min_{X} \quad F(X)$$

$$\textit{subject to} \quad A \cdot X \le B,\ Aeq \cdot X = Beq \quad (\text{linear constraints})$$

$$C(X) \le 0,\ Ceq(X) = 0 \quad (\text{nonlinear constraints})$$

$$LB \le X \le UB \quad (\text{bounds})$$

MATLAB provides a graphical user interface optimtool. We create two files for the objective function and the constraints respectively. We enter *optimtool* in the command window, and put the objective function and the constraints files in the form (Figure 4.4).

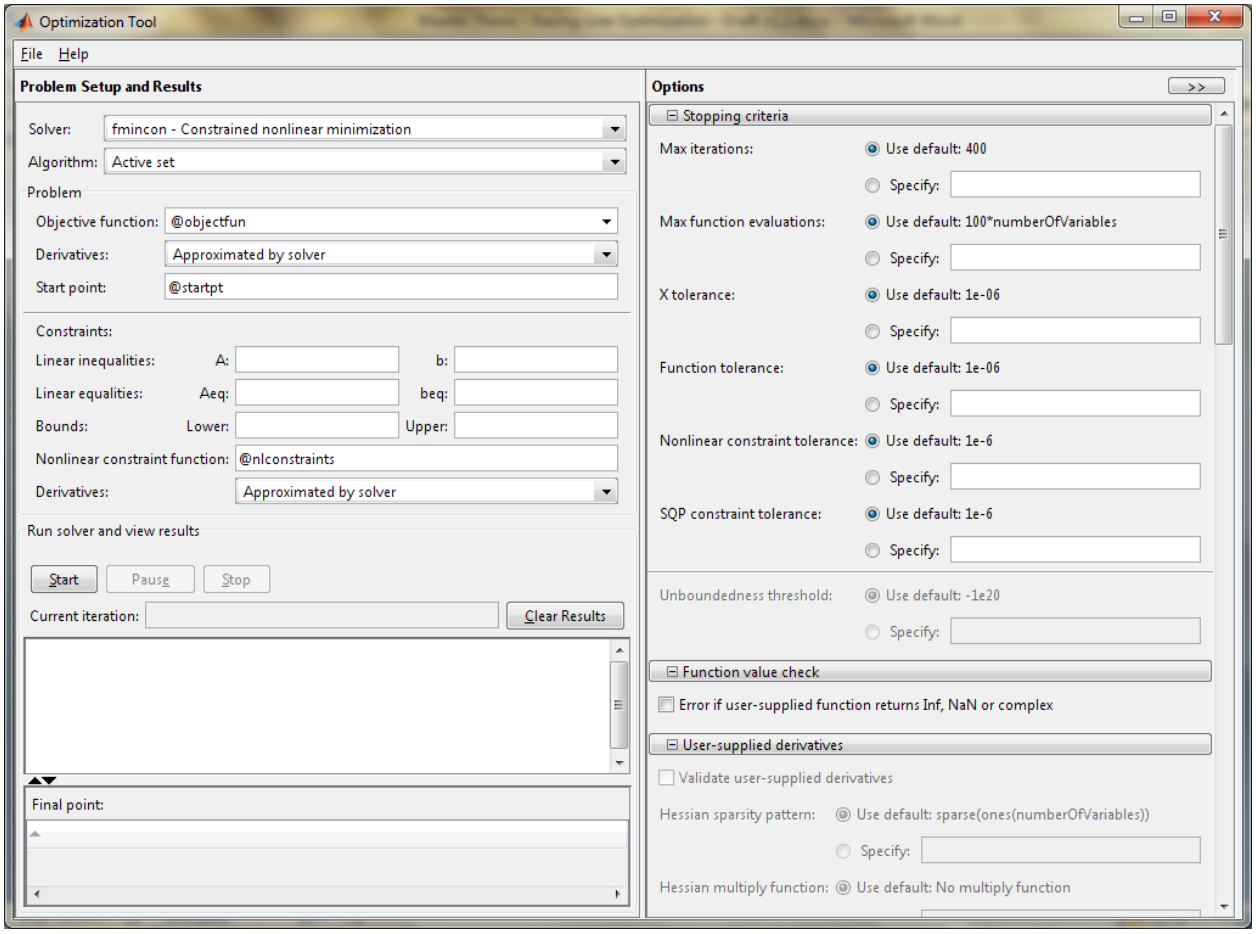


Figure 4.4. Using MATLAB Optimization Toolbox to solve the nonlinear programming problem.

The FMINCON function works well for a small number of points, but when the number of point is too large, it will get stuck in a local optimum and give unreasonable results.

5. Artificial intelligence approach

To make the problem more generalized for solving the problem of a racing car driver, here we are also going to consider the following whole racing circuit track optimization problem.

5.1 The artificial intelligence algorithm for finding optimal racing lines

Our main goal in the artificial intelligence method is to develop an intelligent agent, which is a system that perceives its environment and takes actions that maximize its chances of success. Our method is outlined below:

(1) Set the starting point 0.

(2) For each point i, decide the point i+1 using intelligent algorithm with probability distribution involved.

(3) Run many times and compare. Best energy racing line and best time cost racing line are chosen. Here we tried the "best energy" in the following formulations $\int k^{0.2} ds$, $\int \sqrt{k} ds$, $\int k ds$ and $\int k^2 ds$.

There are two major algorithms in this method: the optimal racing line determination and the velocity determination.

When looking for the optimal racing line, we are trying to make the program smart by carefully trimming possible branches and skewing to the appropriate sides. Let's assume that we use all discrete points first. If there are m points along the center line to express the track (triangles in the figure below), and for each point on the center line there are n possible positions within the width of the track, then connecting from the start to the end there are n^m different possible racing lines. Obviously n^m is exponentially increasing, which means when the track grows a little longer, or the points are made denser along the line, the time cost will very significantly increase. Hence a complete enumeration of all possibilities is not a practical way of solving the problem.

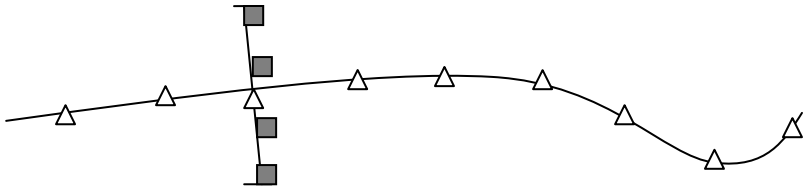


Figure 5.1. Explanation of discrete point picking in searching for the optimal racing line

The basic concept of branch-trimming is to choose options that are more likely to lead to the optimal result, and abandon the options that are insensible or impossible to be an optimum. Besides, purely connecting discrete points may lead to non-smooth lines (piece-wise linear), so we make the selection along the width line continuous (Figure 5.2). How much the probability distribution is skewed will depend on the past steps and the future track shape (Figure 5.3).

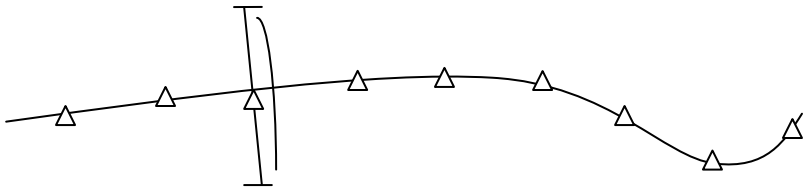
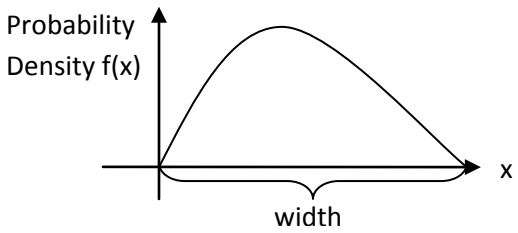


Figure 5.2. Explanation of continuous point picking in searching for the optimal racing line

For each point along the line, the probability distribution of possible positions along the width of the track is illustrated in Figure 5.3. It is skewed according to previous steps taken and future road conditions.



(a)

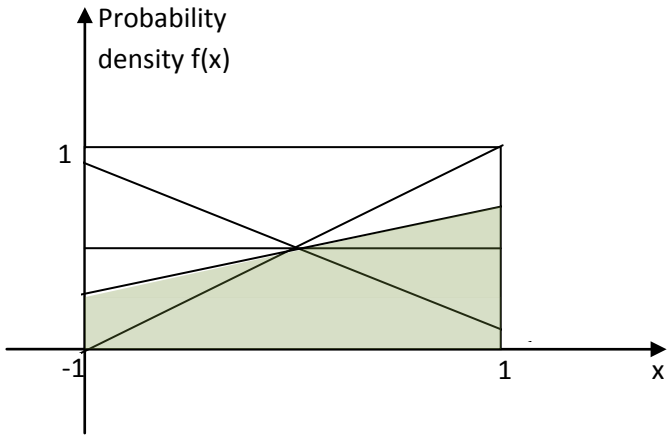


Figure 5.3. Probability distribution of how much point i+1 is skewed from point i. (a) is a skewed normal distribution, while (b) is a skewed linear shape distribution which always passes through the center point (0, 0.5)

One important feature of this intelligent method is that it can apply branch-cutting to largely eliminate possibilities and avoid complete enumeration in a wise way. The branch cutting can be applied easier when we assume there are only several candidates to choose for each point as shown in Figure 5.1. However, when there is an infinite number of candidates for each point along the racing line as in Figure 5.2, the solution is to control the probability distribution of the chosen point. Figure 5.3 shows two possible ways of controlling the probability distribution.

Figure 5.3(a) is a skewed Gaussian distribution. Its probability density function is the Gaussian function $f(x)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{\frac{(x-\mu)^2}{2\sigma^2}}$. When it is skewed to the right side, the point is more likely to fall on the right side. However, this distribution is too dense near its center and it has a large possibility of only staying around the region that the center of the probability distribution defines.

Figure 5.3(b) is skewed more than (a), and its probability density function has a much simpler form $f=ax+b$. The shape of this density function indicates the likelihood for it to result in a wider range of possibilities being tried. The trials will not be biased too much and only dense at the center. It has enough freedom to jump toward either of the two sides.

Some features of our method are explained below:

a) Randomness is introduced when searching for the best line. Randomness is necessary in order to cover a wide range of possible paths.

b) Rules are used to trim branches. For example, within three consecutive points, point 2 and point 3 should not bend too much toward different directions. The more point 2 bends to the left from point 1, the less point 3 is allowed to bend to the right of point 2. This is explained in Figure 5.4 below.

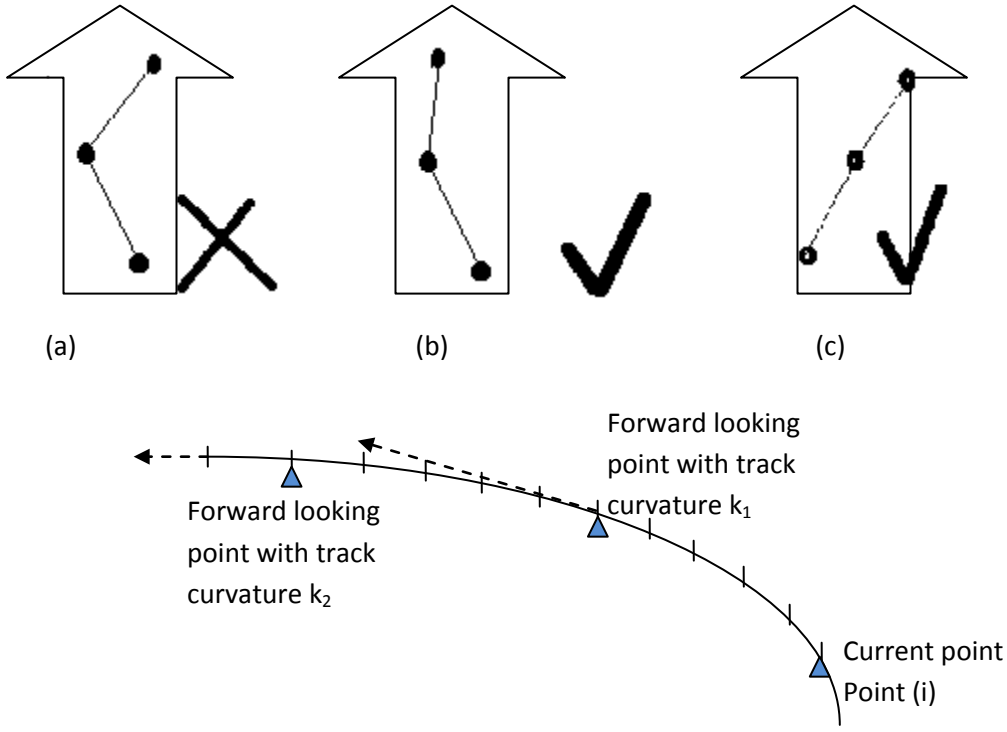


Figure 5.4. No jumping to different sides too much on adjacent points constraint. Assume that the big arrow shows the direction of the step previous to all 3 points. (a) is not allowed because it turns left from the 1st point to the 2nd point and wants to turn right too much at the 2nd point. (b) and (c) are allowed because they did not jump too much to different sides at two consecutive points. (d) gives a simple example of the forward-backward looking.

Let's take a look at Figure 5.4(d). Assume that k_1 and k_2 are the signed curvatures of the center line of the track, and let $MaxK$ be the maximum absolute value of curvature along the whole racing track. The number of points we look ahead is such that, between point i and the point where k_1 is calculated, no more than a 45 degree turn of the track is completed. Define

$$s = \min\left(\max\left(\frac{k_2 - k_1}{MaxK}, -1 \right), 1 \right)$$

Set the probability distribution function as

$$f(x) = \frac{1}{2} + \frac{1}{2} s \cdot x$$

For the randomness, pick a random number p which satisfies $p \in [0,1]$. Solve the equation

$$F(x) = \int_{-1}^{x} f(t)dt = p$$

for the variable x. As the integral of $f(x)$ is easy to calculate mathematically we have

$$sx^2 + 2x + (2 - s - 4p) = 0$$

Thus $x = \frac{-1 \pm \sqrt{1 - s(2 - s - 4p)}}{s}$. Because x is somewhere between -1 and 1, only the positive sign will be taken no matter if $s > 0$ or $s < 0$.

For three-dimensional tracks, the looking ahead algorithm is the same except that it uses the geodesic curvature of the center line instead of the absolute curvature.

With this algorithm, when there is a corner not far in front, the look-ahead function will notice that and let the car have larger probability to lean to the outside edge of the track to allow a larger radius for making the turn and hitting an apex at an appropriate point.

From the experiments, we find that $\int \sqrt{k} ds$ is the better measure and with this definition of E, the best E and best time cost racing lines are almost the same. This can be explained theoretically: since $k_{\max} = 0.5$, all curvatures along the line are less than 0.5. So for curvature changes, $\sqrt{k}$ is usually more sensitive than k^2; thus it reflects the curving of the racing line in a more amplified way and leads to smoother and generally better results. In the experiments, we have tested using different exponents for k and find that $\int k^{0.5} ds$ works best in this situation. More results are discussed in Chapter 5.2.

The artificial intelligence approach can be used for any shape of racing tracks no matter if they are 2-dimensional or 3-dimensional. It makes smart decisions on what points to take to minimize the time cost.

MATLAB code is used to generate 2-D or 3-D racing tracks. The artificial intelligence racing line finder is implemented in C++. We also write a display program for all shapes of tracks in Visual Basic.

Define E as $E = \int_s \sqrt{|k(s)|}ds$. Here we will show the two results from the AI approach which lead to optimal time and optimal E, respectively.

In the program, we keep track of the two racing lines: the line that has the smallest time cost, and the line that has smallest value of E. The E-best racing line is chosen according to its E value instead of its time cost, and that is why the E best racing line always costs at least as much time best racing line. In the plots, we call E *energy*, but be noted that here "energy" refers to the integral of square root of curvature, not the physical energy. The flow chart of the program is shown in Figure 5.5.

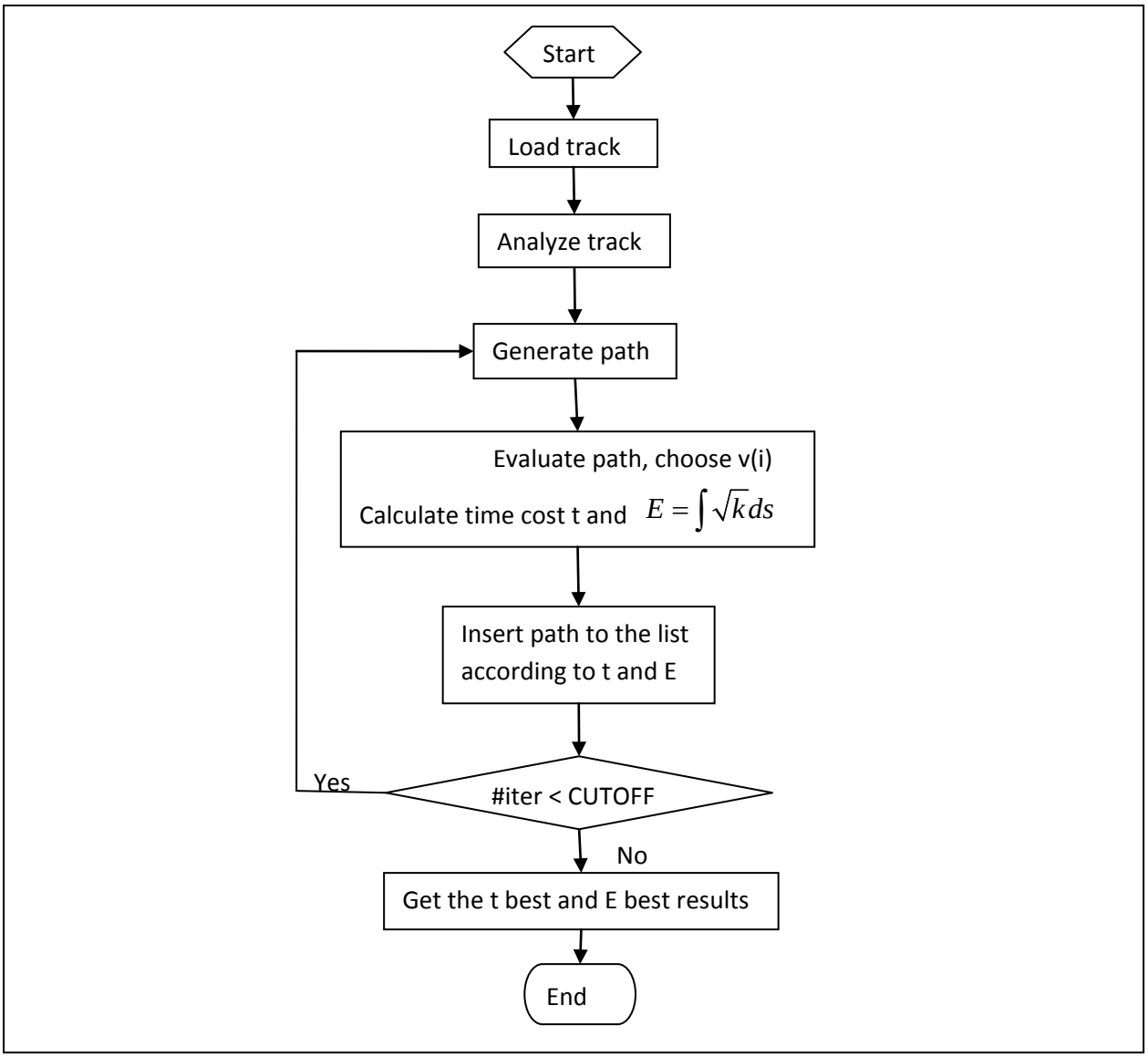


Figure 5.5. Flow chart for finding the optimal racing line

The method is based on intelligently random tests. When we generate the path, branch cutting is used to reduce the number of iterations needed to achieve an optimal solution. Suppose that we start from the center line at point 0. The distance from each point to the center line will be somewhere between (-d/2, d/2). From point 1 and onwards, the range of distance to centerline depends on the previous steps to avoid too much zig-zagging. For example, if the previous step is leaning to the right, the next step cannot lean to the left too much. The looking ahead technique is also used to see how long of a straight line is left or how far the next corner is. The probability of the point falling on some position relative to the center line will be skewed appropriately.

The procedure to choose velocity $v(i)$ step by step for a given path is as follows

1) At point 0, the velocity is initialized to 0, i.e., $v(0)=0$
2) For $v(i)$, firstly calculate the $v_{\max}$ without considering the power constraint. If $v_{\max}$ satisfies the deceleration constraint, then choose $v(i)$ based on $v_{\max}$ and the power constraint. If not, set $v(i)=v_{\max}$ and correct $v(j), j<i$ from $j=i-1$ .Correct $v(i-1)$ to satisfy the maximum deceleration constraint, and if necessary, correct $v(i-2), etc.$

This procedure is explained in Figure 5.6 below.

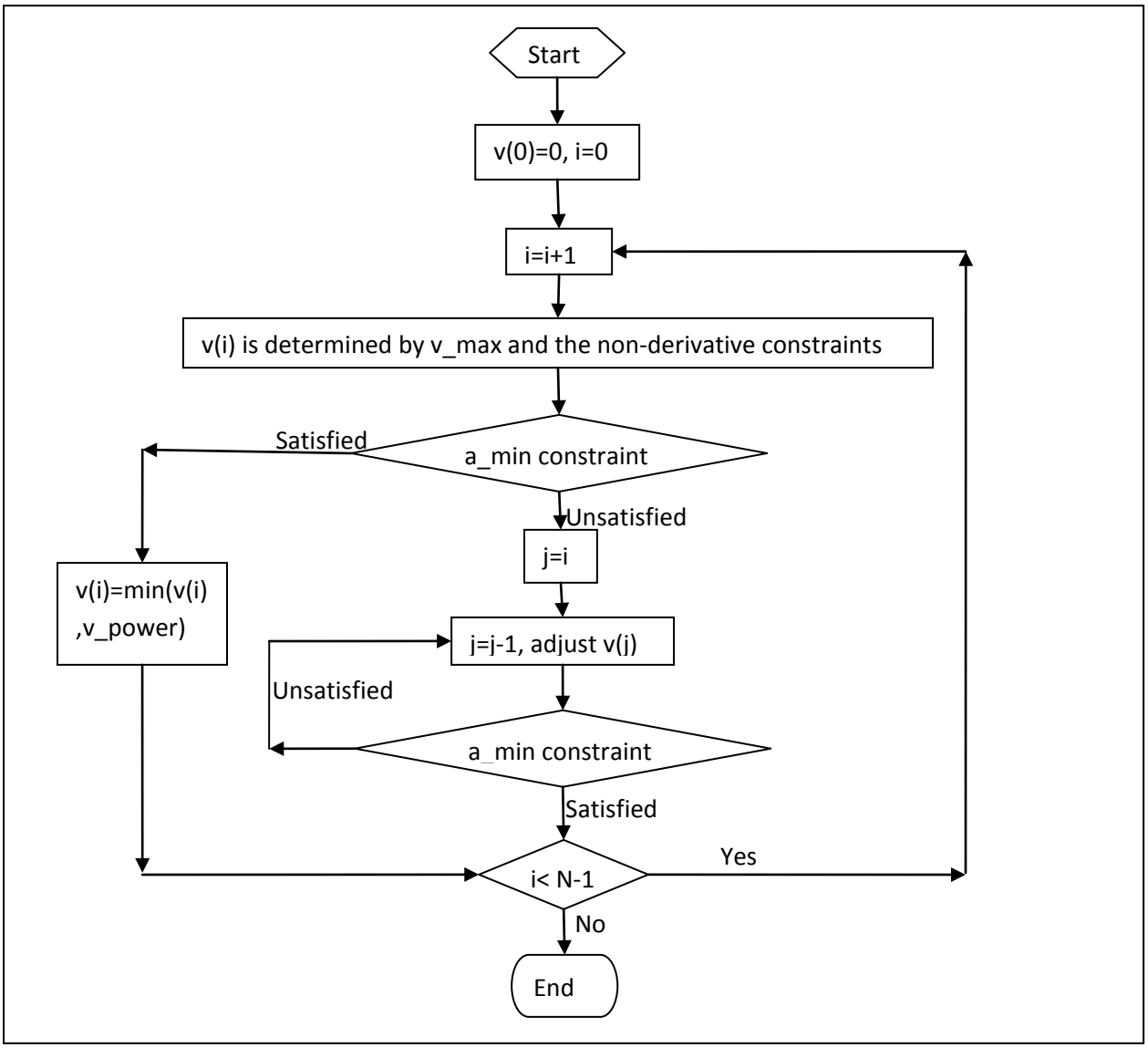


Figure 5.6. Flow chart for finding the velocity along a racing line

Let's analyze the time complexity of the artificial intelligence algorithm. Assume that the number of iterations is m, and the number of points to describe the track is n. Because of the backwards adjustment steps, the worst time cost will be $O\left(m \cdot n^2\right)$. If we see m as a fixed number instead of a variable, the time complexity is $O\left(n^2\right)$. The best situation would be that no backwards velocity adjustment is necessary, and the time cost will be only $O\left(m \cdot n\right)$, or $O\left(n\right)$ when m is fixed.

To improve the performance of the algorithm, a divide-and-conquer strategy of "big-small step" is also introduced. In this strategy, the points will not be determined one by one. Instead, it will be once every n points, where n is the number of small steps in a big step. The step size of a big step is n times the step size of a small step. This strategy will expose the program to more possible options at first, and then choose remaining points based on the optimal solution of the first stage results. This method turns out to be very effective in our experiments. The results analysis after applying this improvement algorithm is in Chapter 5.2.3.

5.2 Implementation and results

We implemented the artificial intelligence method for both 2-D and 3-D racing tracks, and the results are obtained and studied.

5.2.1 Optimal racing line for 2-D racing tracks

We can get the optimal racing lines according to any given 2-dimensonal racing tracks and car information. The 2-dimensonal situation is a simpler version of the 3-dimensonal situation. Three kinds of tracks are tested out here.

5.2.1.1 Flower shape racing track

The flower shape racing track contains 2 symmetric three-quarter-circles, a half circle and a straight line. In the following plots, the outside edge line and the inside edge line (both in blue color) are used to represent the racing track and the optimal racing line found is represented by the red line.

(1)30,000 iterations

The comparison of E best and t best results are listed in Table 5.1.

Table 5.1. Comparison of E best path and time best path after 30,000 iterations of the Artificial Intelligence method for Flower Track

	E best	Time best
E ($E = \int_s \sqrt{\lvert k(s) \rvert} ds$)	288.843	291.369
Time cost (sec)	124.283	123.735

The optimal racing line generated is displayed in Figure 5.7.

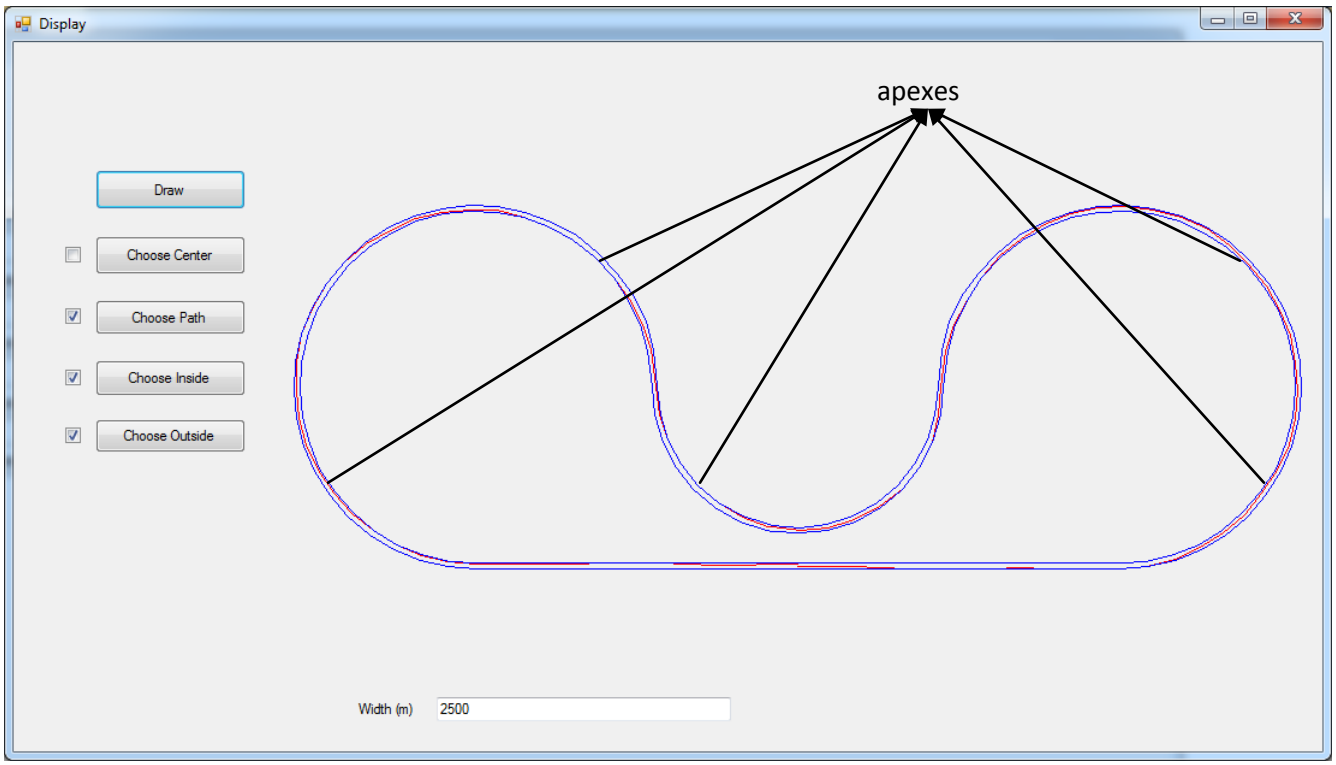


Figure 5.7. Optimal racing line for flower track using AI method after 30,000 iterations. The apexes are marked.

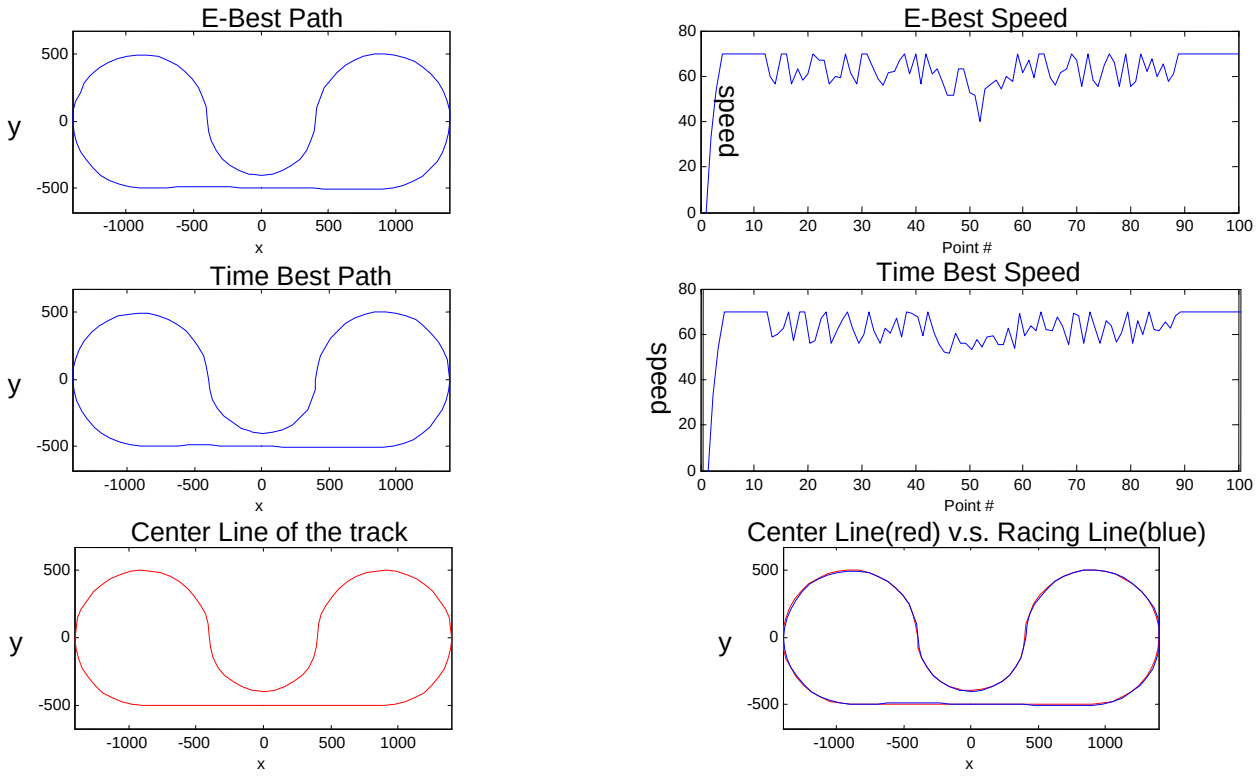


Figure 5.8. Breakdown analysis graphs for results after 30,000 iterations

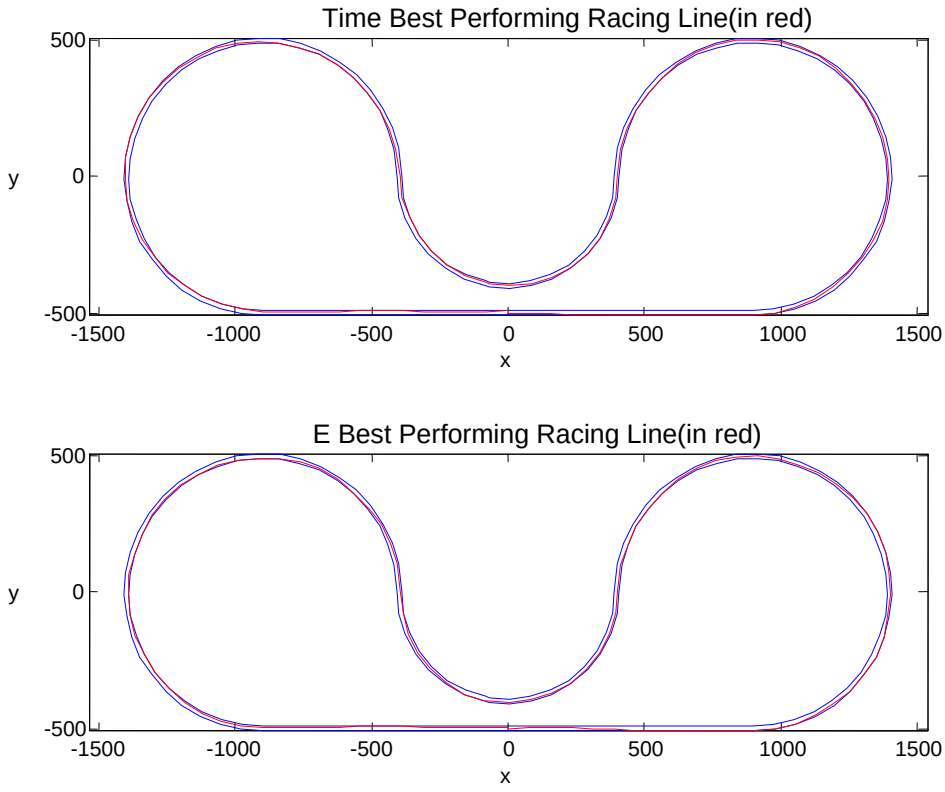


Figure 5.9. Racing lines of E best and time best after 30,000 iterations

It is interesting to notice the difference between time best path and E best path. In Figure 5.9 shown above, the E-best path tends to stay around the outside line more, which is understandable because its main goal is to achieve the lowest curvature. Besides, the time best path has a smoother speed graph as shown in Figure 5.8.

(2)100,000 iterations

When we increase the number of iterations from 30,000 to 100,000, the results are improved slightly (0.038% in this case).

Table 5.2. Comparison of E best path and time best path after 100,000 iterations of the Artificial Intelligence method for Flower Track

	E best	Time best
E($E = \int_s \sqrt{\lvert k(s) \rvert} ds$)	288.515	294.566
Time cost (sec)	124.643	123.688

Comparing Tables 5.1 and Table 5.2, we can see that the time cost for the time best path becomes a little smaller, and the E value for the E best path becomes a little smaller. The paths and speed graphs of the E best line and time best line are displayed in Figure 5.10.

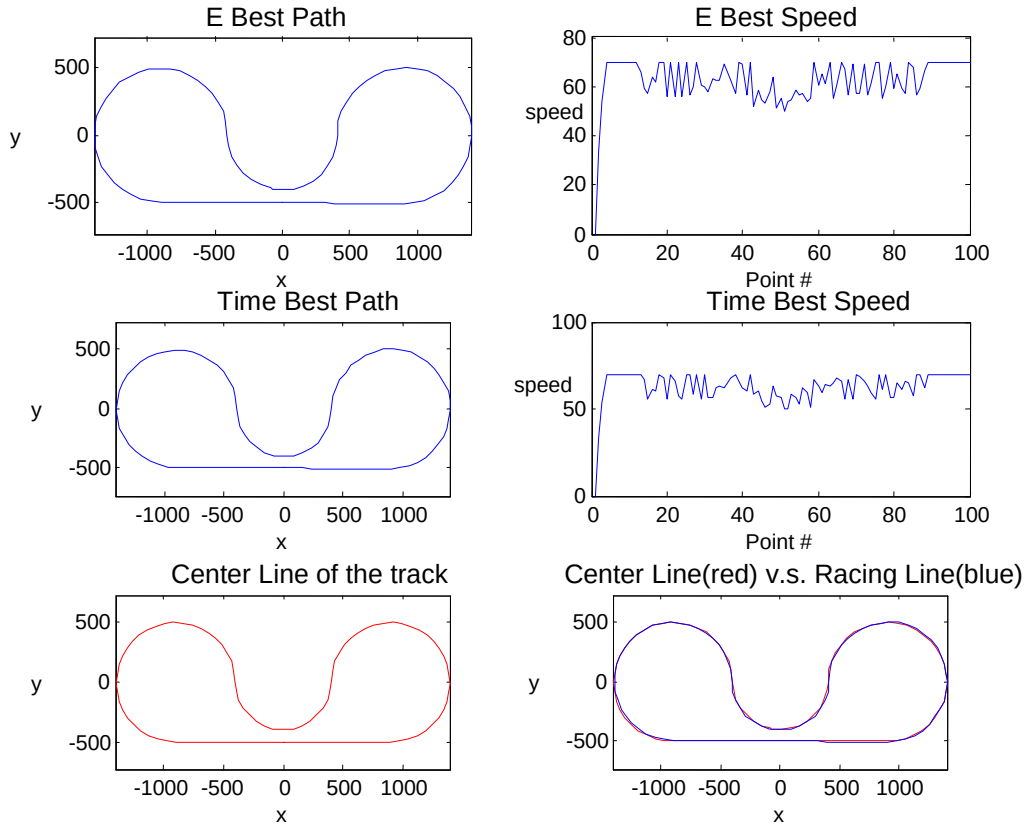


Figure 5.10. Breakdown analysis graphs for results after 100,000 iterations

From Figure 5.10, we can see again that the speed graph for the time best racing line is much better than the E best racing line; the car can generally stay more at the high speed level. The E best racing line may sacrifice the real time cost for making the $\int k^n ds$ value smaller.

The larger graphs showing the time best racing line and the E best racing line are in Figure 5.11.

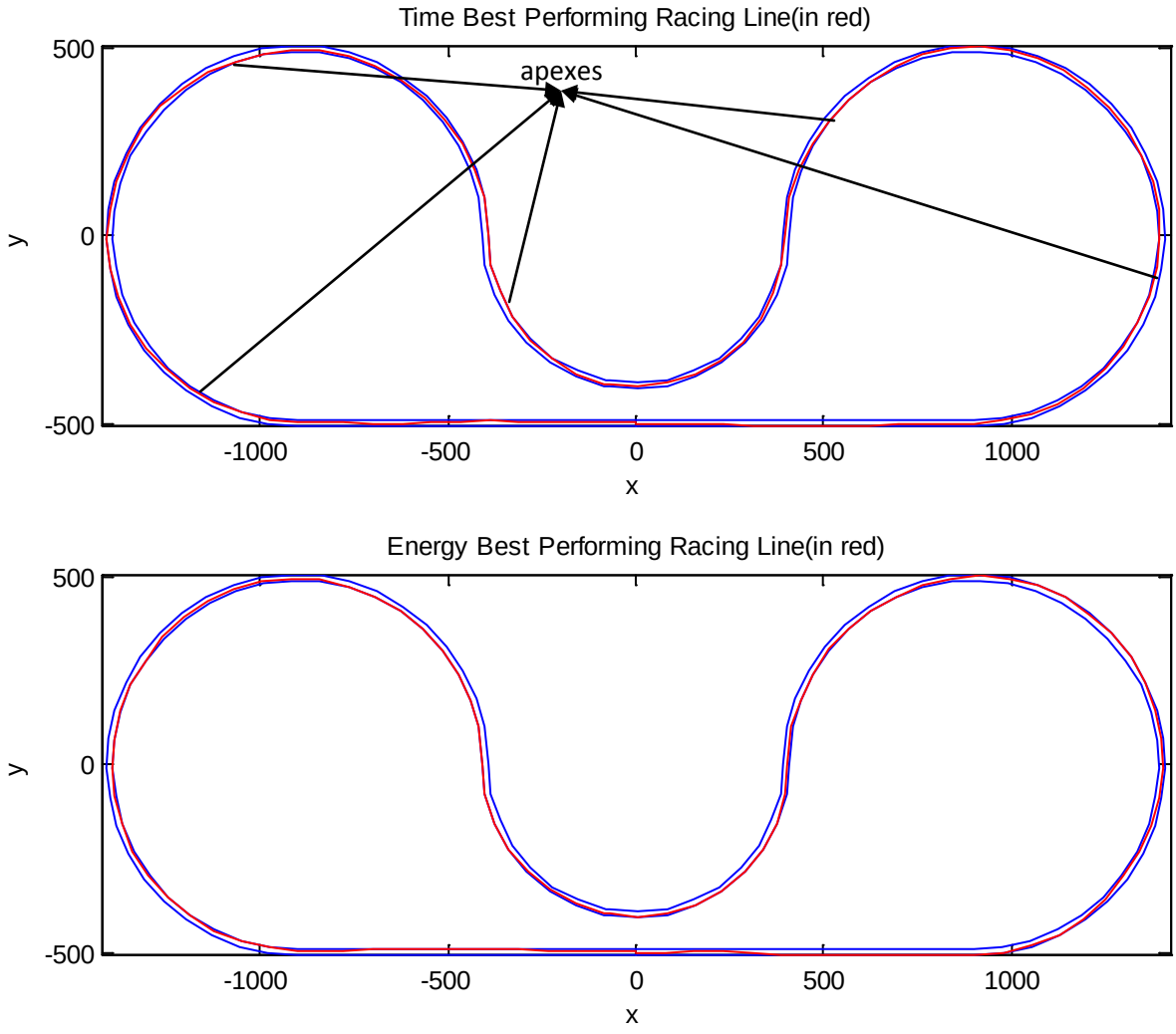


Figure 5.11. Time best and E best racing lines after 100,000 iterations. The apexes in the time best racing line are marked.

From the results, we can see that with more loops the results could be better, but 30,000 loops are already enough to get a reasonably good answer. Besides, due to the probability factors in the program, more loops do not always guarantee better results – they just usually get more precise results.

5.2.1.2 Soccer field racing track

Next, we use MATLAB to generate a track consisting of two double circles and two straight lines connecting them. According to the experimental results, after trying several n values for E = $\int k^n ds$ we get the best result here by setting E to $\int_s \left[k(s)\right]^{0.2} ds$

The time cost results are in Table 5.3 below.

Table 5.3. Comparison of E best path and time best path after 30,000 iterations of the Artificial Intelligence method for Soccer Field Track

	Time	E
Time best (sec)	**78.6361**	811.397
E ($\int_s \left[k(s)\right]^{0.2} ds$) best	79.1217	685.408

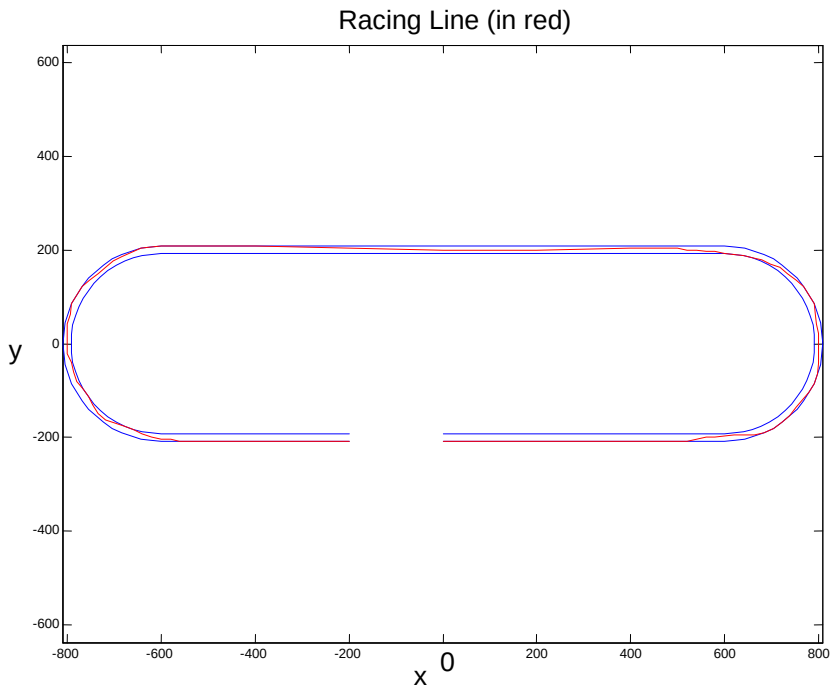


Figure 5.12. MATLAB plot of the optimal racing line generated by the Artificial Intelligence method for the soccer field shape track

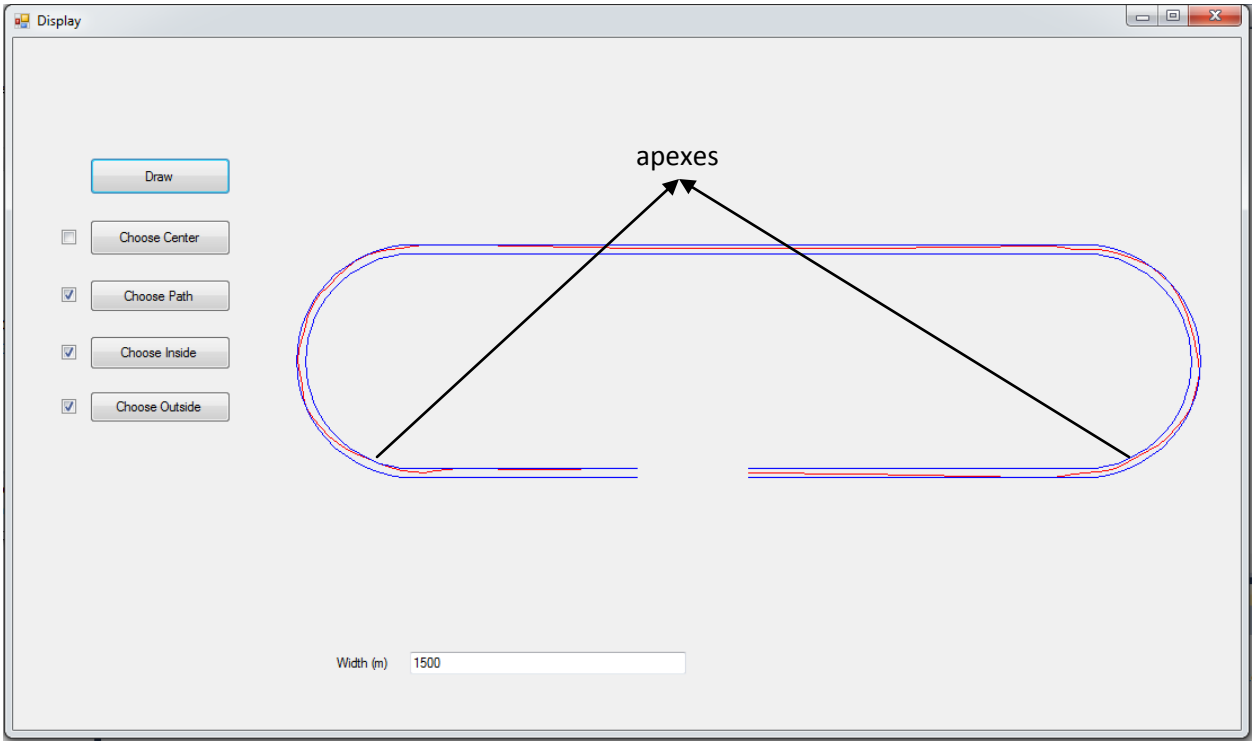


Figure 5.13. Display software written in VB. It draws the results of the Artificial Intelligence method for the soccer field track. The two apexes are marked.

We can see there are two apexes in the optimal racing line, one for each corner.

5.2.1.3 Ellipse racing track

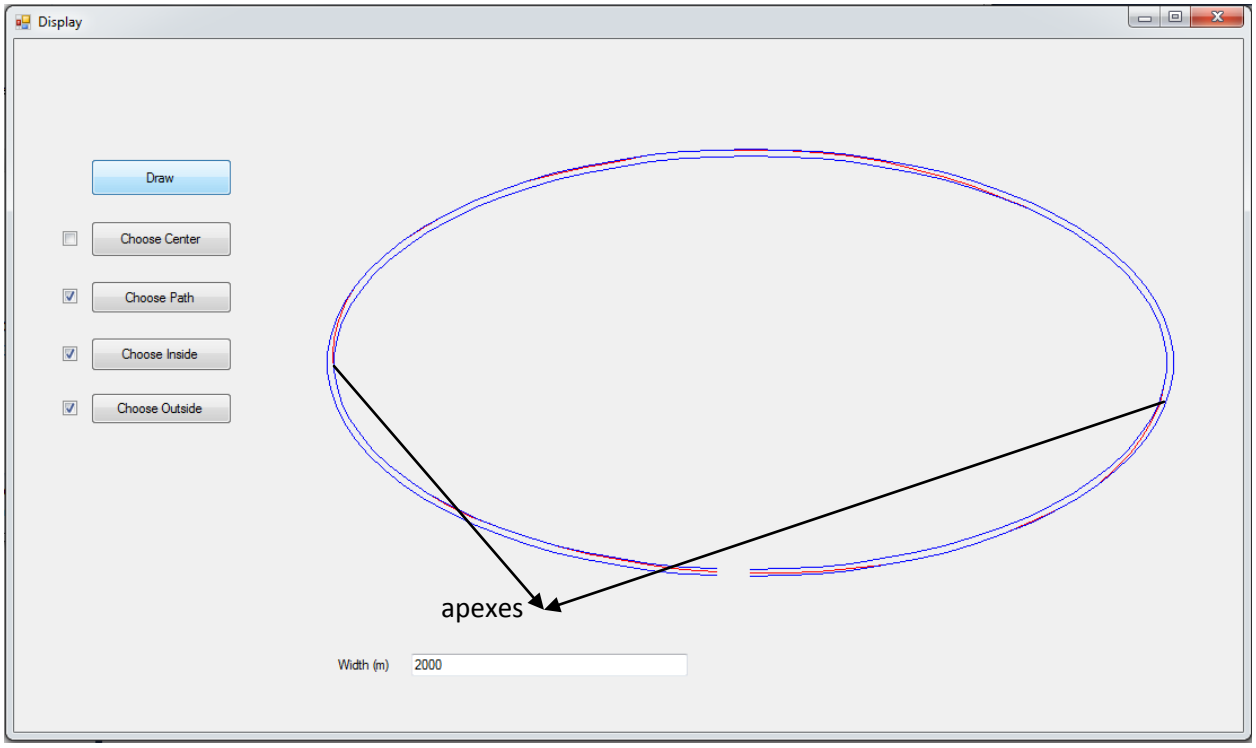


Figure 5.14. Display software drawing the results of the Artificial Intelligence method for the ellipse track. The apexes are marked.

The ellipse track is different than the other tracks that we tested in that it does not have any straight parts. The whole track has some curvature; thus it will limit the speed of the car and require the racer to make smart and consistent decisions along the whole track. From the results above, we can see that there are two points that can be called apexes, but the concept of apex is not as concrete. Overall, the optimal racing line makes the ellipse rounder.

5.2.2 Optimal racing line for 3-D racing tracks

The three-dimensional track is more likely to appear in real life racing. To observe the result, we introduce a new method here. The display application is written in Visual Basic.

In two-dimensional situations, we only need to see the whole race track in one plane, but in three-dimensional situations it may have different views from different angles. Assume that there is a photographer taking the photo of the track, then the photo he takes will depend on where he stands (x, y and z coordinates), the angle that the camera is pointing from (theta and phi), as well as the features of the camera (width and focus). This is illustrated in Figure 5.14.

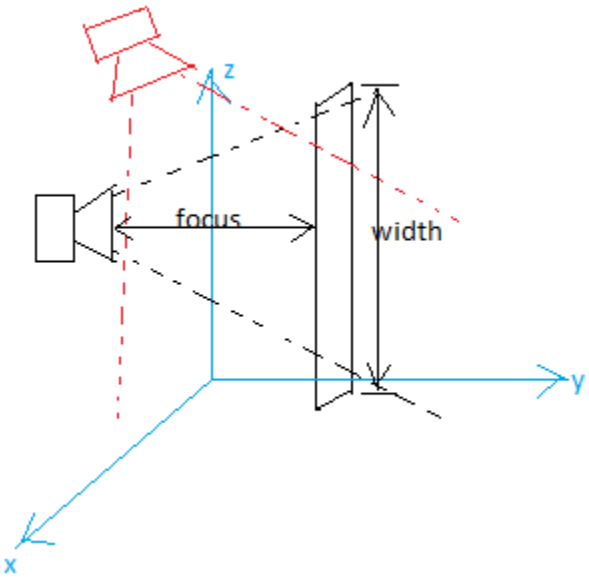


Figure 5.15. Looking at the racing track behind a camera. The blue lines show the x, y, and z axes; the black lines show the camera and its focus and width of taking views; the red lines show another possible position and direction of the camera.

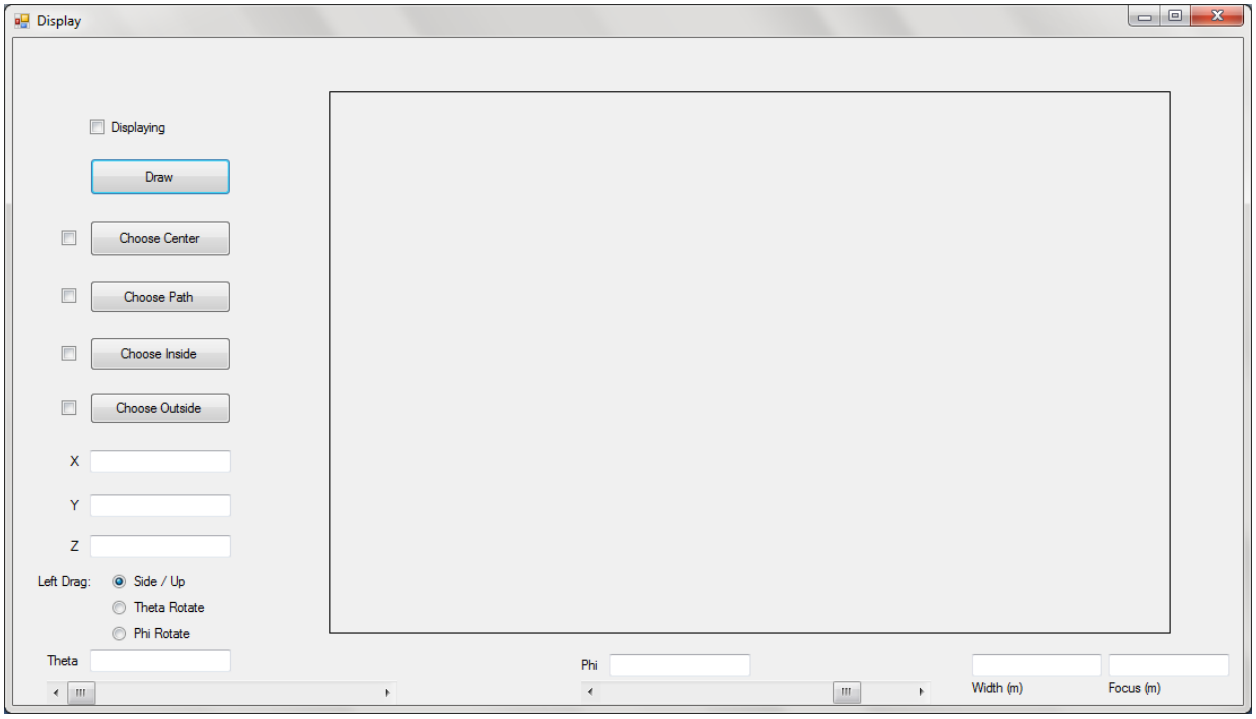


Figure 5.16. The new modified software interface to display racing lines on 3-D racing tracks

Tick in the boxes that you want to display. For example, if you want the optimal path found by the program to be displayed, tick the box and click on the “Choose Path” button to choose a path file. If you do not want to show the center line, do not tick the box in front of the center line button.

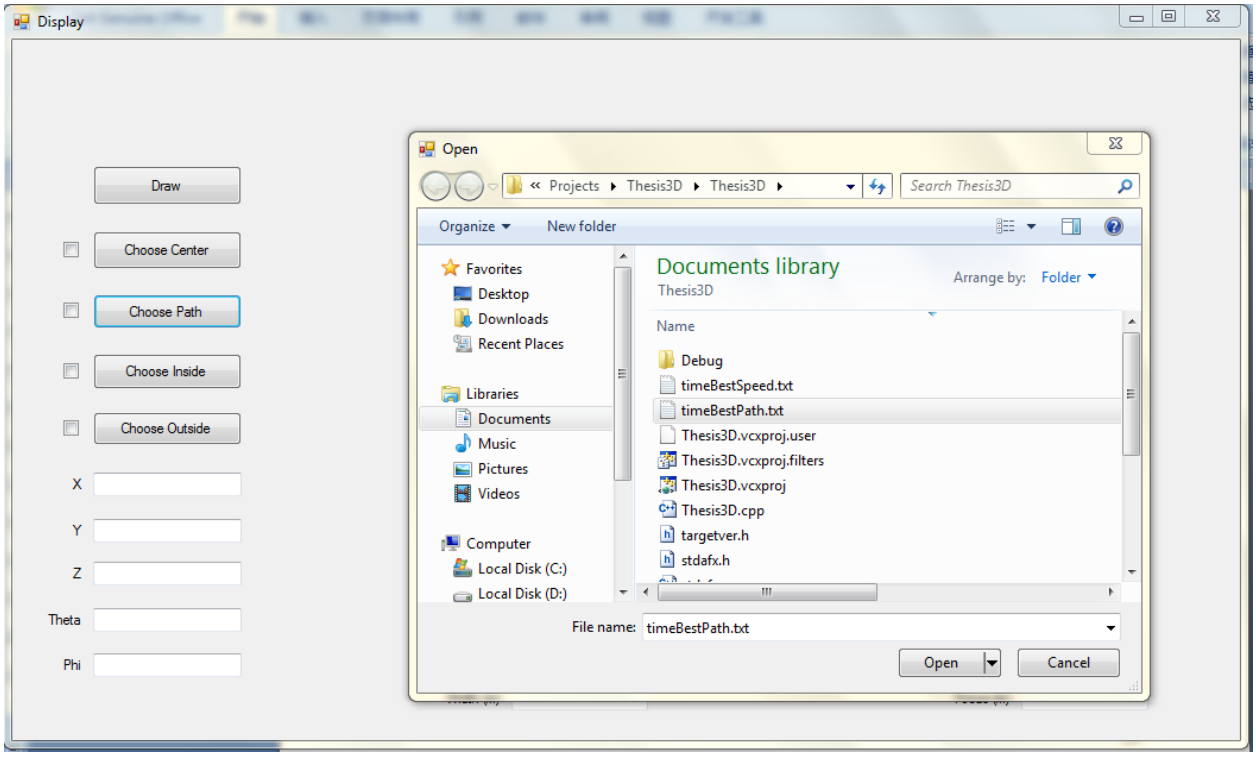


Figure 5.17. Choose path in the 3-D display program. The 3-D display software is similar to the 2-D display software. Select files of the lines that we want to display and click draw. The only difference from 2-D is that we can adjust the values of x, y, z, theta, phi, width and focus as explained before in Figure 5.14.

After choosing files for the path line, inside line and outside line, set the position, direction and features of the camera. The origin point (0,0,0) is the center point of the whole soccer shaped racing field. In Figure 5.18 the camera is at the southwest corner and high up, pointing towards the field.

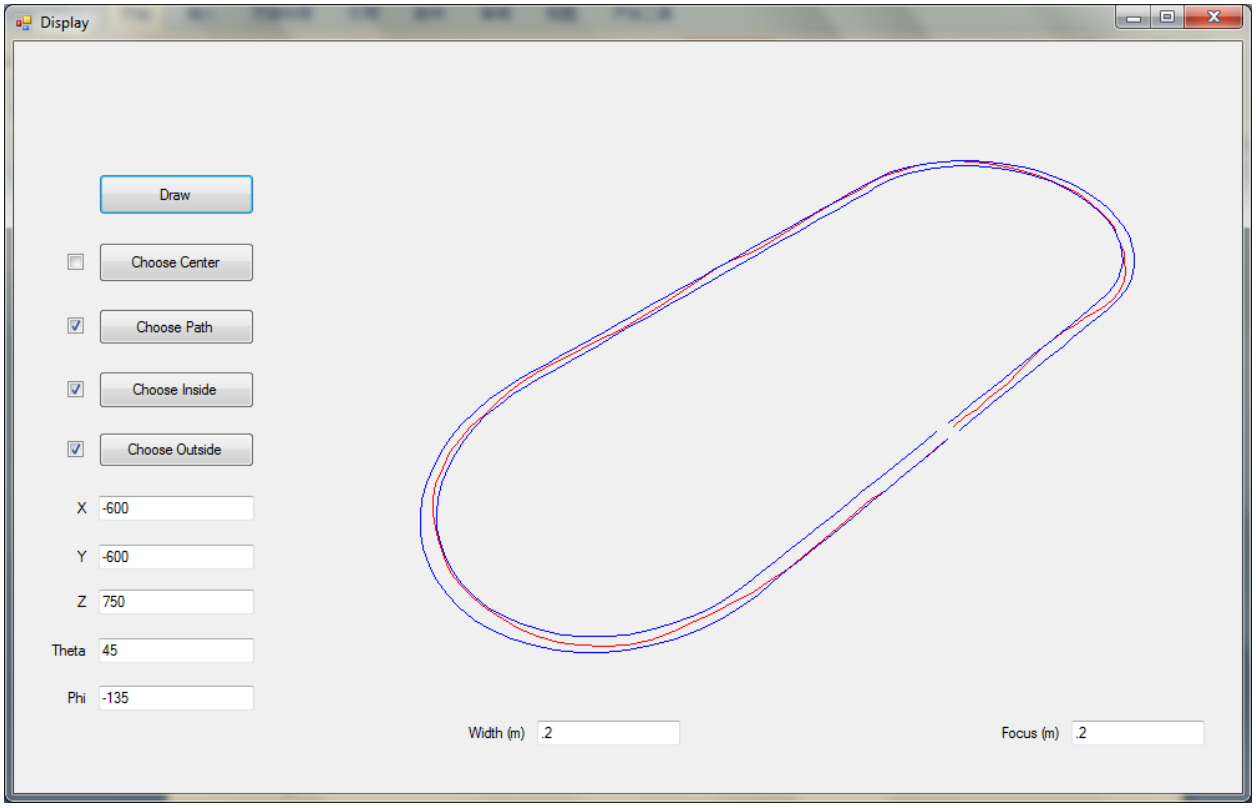


Figure 5.18. Display of optimal racing line for the 2-D soccer field track. We can see the difference between the 2-D and 3-D display softwares here. The angles and distance are adjustable.

One thing to notice is that the display in Figure 5.18 is only the result of a 2-D track using the 3-D display application. This is transitional for introducing some banking to the racing track.

In Figure 5.19, the results for a real three-dimensional track are displayed. The track has banking around both turns (like NASCAR racing tracks), and the results are very interesting to look at. From the same viewing angles and distance as in Figure 5.18, we can see the two late-apexes. But with the banked turns, the cornering is easier and can achieve high speeds. There are adjustments in the two straight line parts to achieve the two late-apexes. Note that for three-dimensional tracks, we define E as

$$E = \int \sqrt{\frac{|k_g|}{n_z - \frac{\operatorname{sgn}(k_g) \cdot n_{car_z}}{\mu}}} ds$$

This is a generalized expression which reduces to $E = \int \sqrt{k}\, ds$ for two-dimensional tracks.

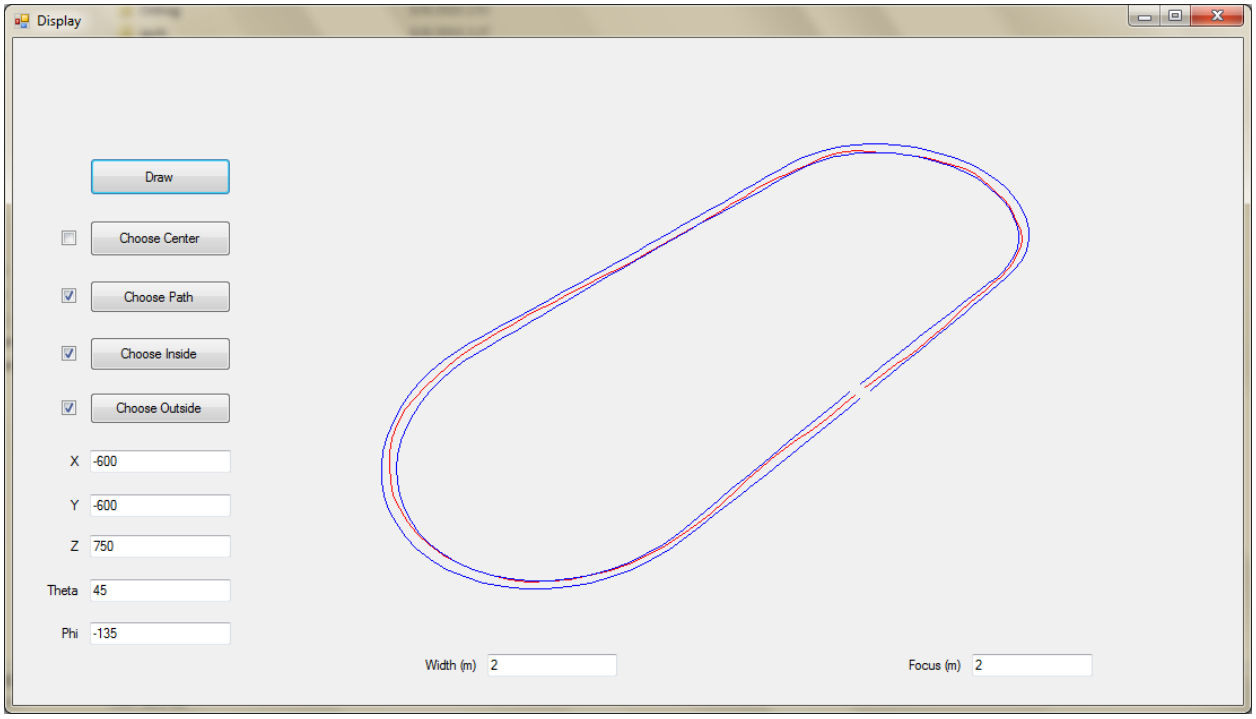


Figure 5.19. Three-dimensional display of optimal racing line in 3-D track with banked corner. The view is from the audience seat high up from the southwest corner

Figure 5.20 gives a clearer view of the banking part around the corner. Now the viewing position is right at the origin point $(0,0,0)$. For two-dimensional racing tracks, the picture taken with the camera placed right at the origin with zero height will just be one straight line. Here in Figure 5.20 we can clearly see the outside line of the track, the inside line of the track and the racing line. Obviously the outside line is higher than the inside line, and this bank can help cars achieve higher speed when they are turning.

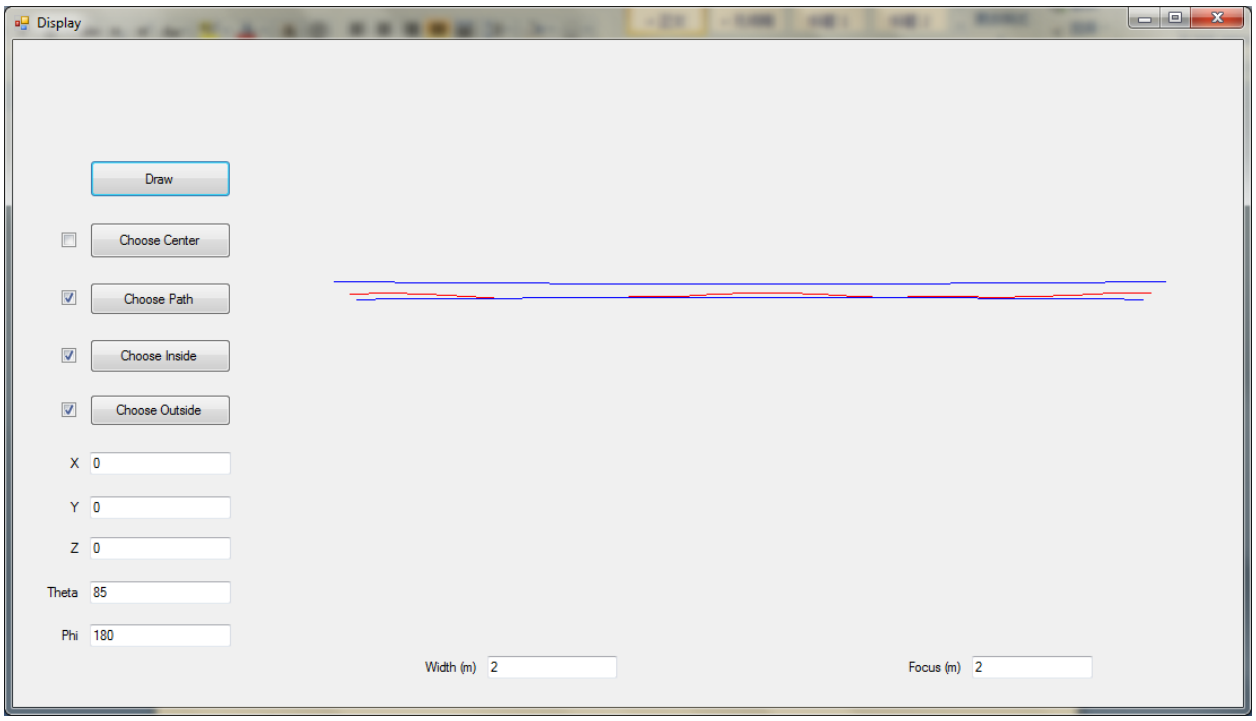


Figure 5.20. Three-dimensional display of optimal racing line in 3-D track with banked corner. The view is from the ground at the center of the soccer field and looking directly to the corner

Its MATLAB plot for the 3-D track is in Figure 5.21. From the graph we can easily see the banking along the track. Here we set the banking angle to be constant. However, it can also be set to change if needed.

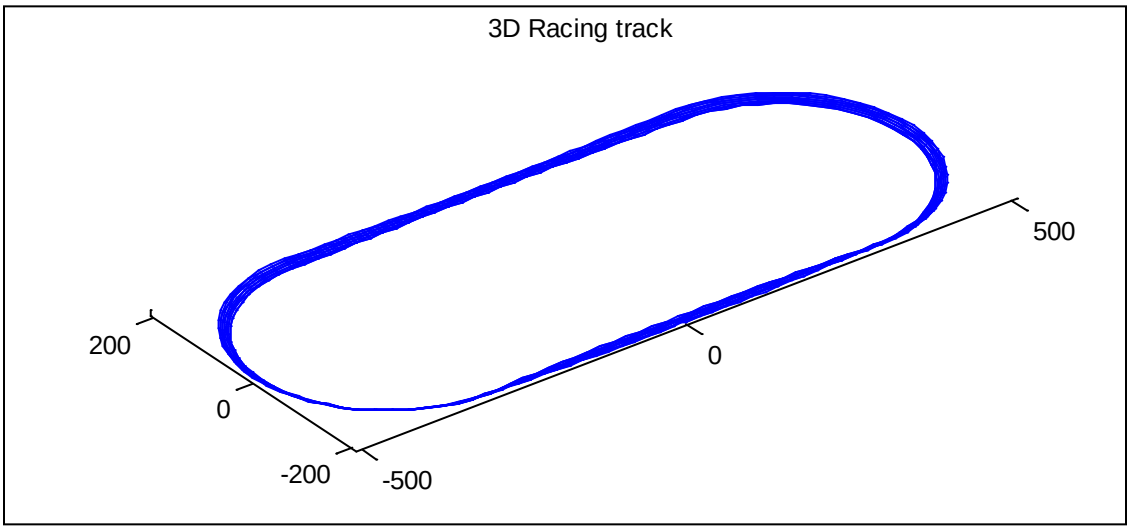


Figure 5.21. MATLAB 3-D plot of the 3-D soccer field racing track. The 3-D track has banked corners. We are viewing it from the audience in the southwest corner.

The optimal racing line on this 3-D track is shown in the *-line of Figure 5.22.

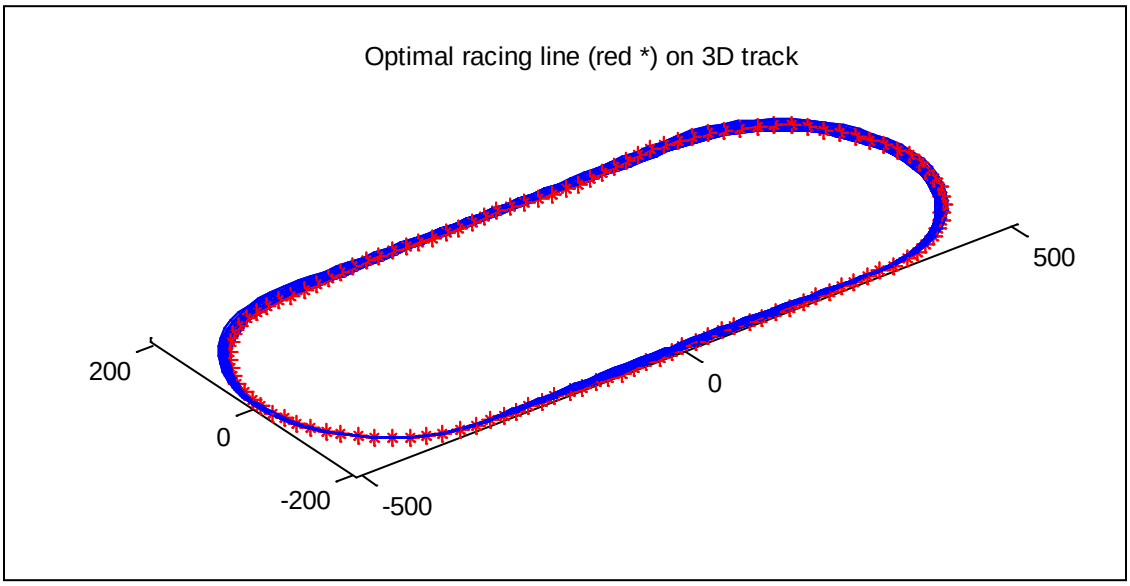


Figure 5.22. MATLAB 3-D plot of the 3-D soccer field racing track. The 3-D track has banked corners. We are viewing it from the audience in the southwest corner.

NASCAR racing tracks have banked corners designed to increase the maximum speed allowed during turning because when the surface of the road leans toward the corner center, there will be a component of gravity that can serve as lateral force for the turn. In our experimental results, the 3-D track also shows its advantage in the overall speed around the corner. The optimal racing line velocity graphs for two tracks are shown below. Figure 5.23 is the track with unbanked corners, and Figure 5.24 has constant banking angle.

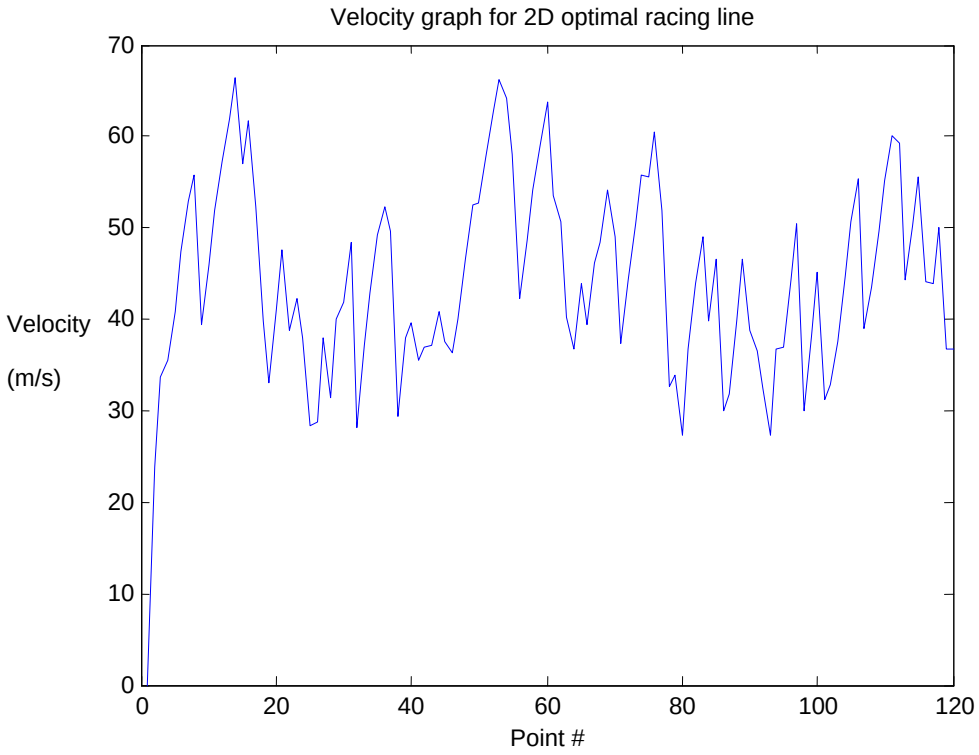


Figure 5.23. Velocity graph for 2D optimal racing line (no banking)

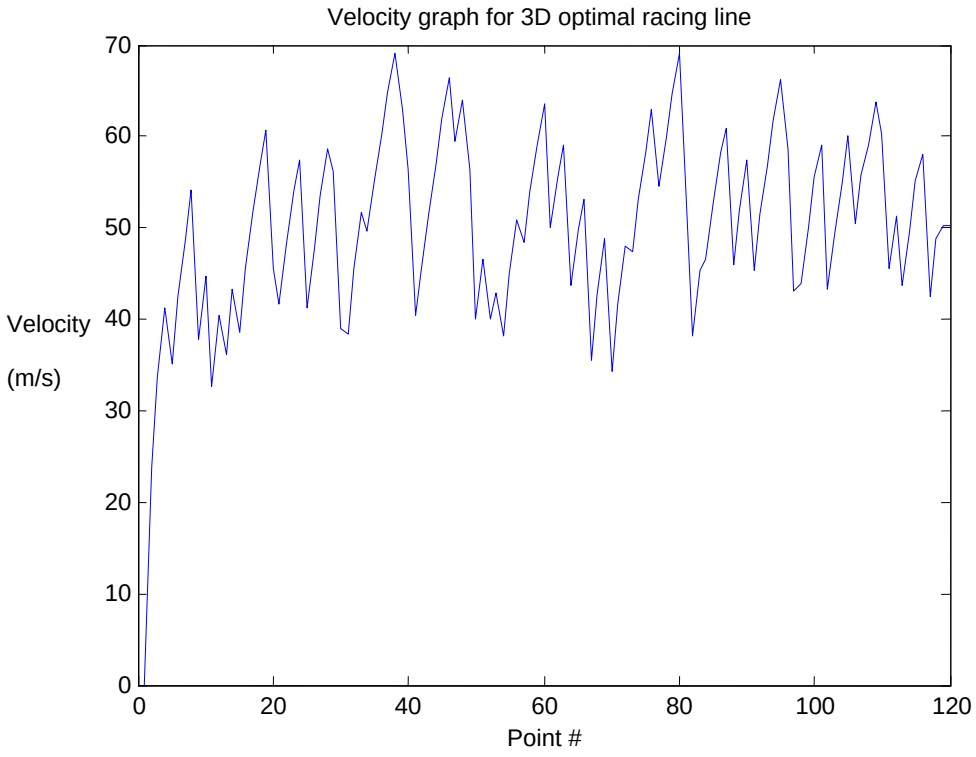


Figure 5.24. Velocity graph for 3D optimal racing line (with banking)

Note that on the x-axis is the point number, and in this soccer-field track case the points are not evenly distributed. The ups and downs of speeds are within the limits of the minimum/maximum accelerations.

The graph below (Figure 5.25) plots the differences between Figure 5.23 and Figure 5.24. As a whole, banking can increase the speed because it makes the maximum cornering speed higher.

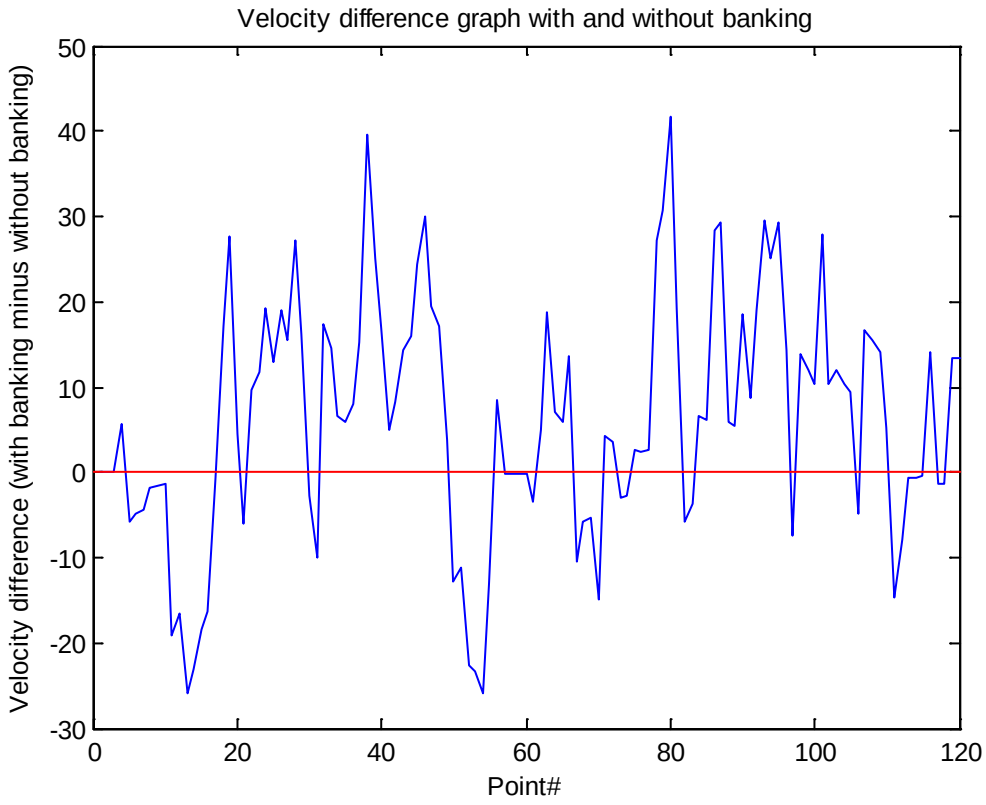


Figure 5.25. Velocity difference graph for two racing tracks with the same center line, one with banking and one without. The points are floating but most of the graph is in the upper half.

The banking part can generally increase the speed of turning if designed properly. The time cost for going through a corner may be decreased dramatically when the corners are banked.

To make the velocity graph smoother, we replaced the upper acceleration constraint with a power constraint as described in Chapter 2.4. The velocity graph after using the power constraint is in Figure 5.26.

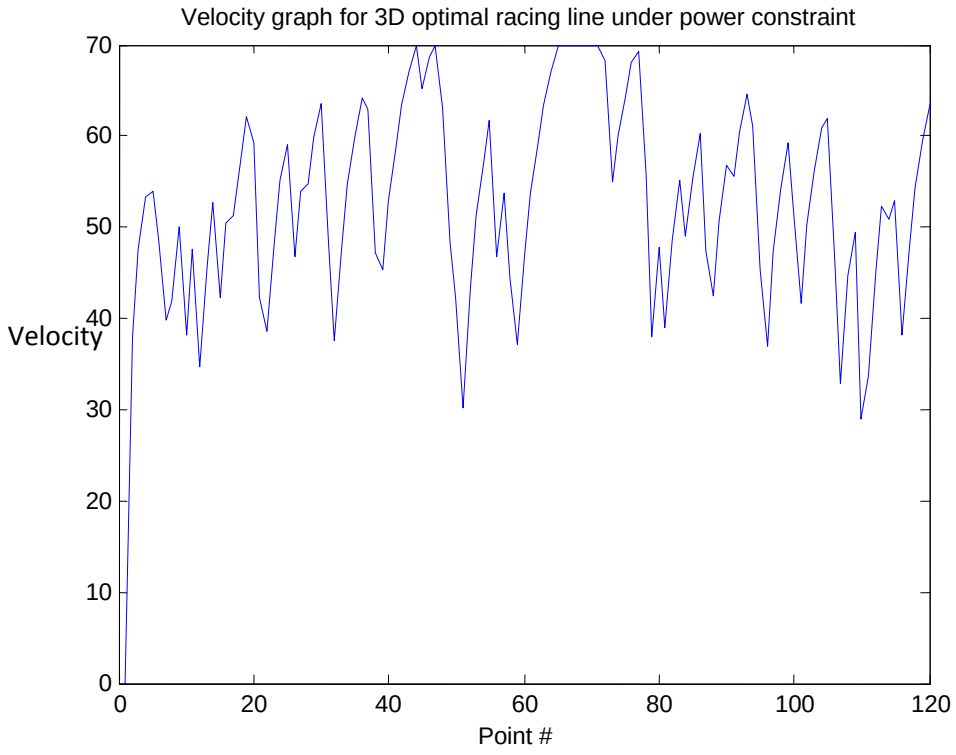


Figure 5.26. Velocity graph for 3D optimal racing line under power constraint. Because of the constraint by power in the speed-up section, the up slope of the velocity graph is smoother than in Figure 5.24.

Let's test the method again on the flower track, and use the power constraint instead of the acceleration constraint. The 3D flower track has banking at all of its corners, and it is flat on the straight line part (Figure 5.27).

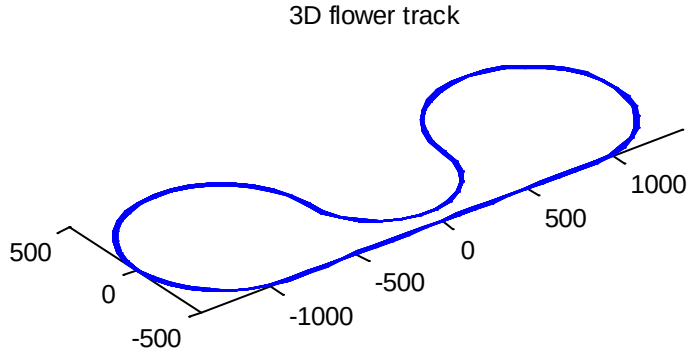


Figure 5.27. MATLAB plot of the 3-D flower track

The track is shown in the graph above. Because the banking angle is only 6 degrees, it is not very obvious to see the three dimensional features, but we can still see it in the change of width from this observation angle.

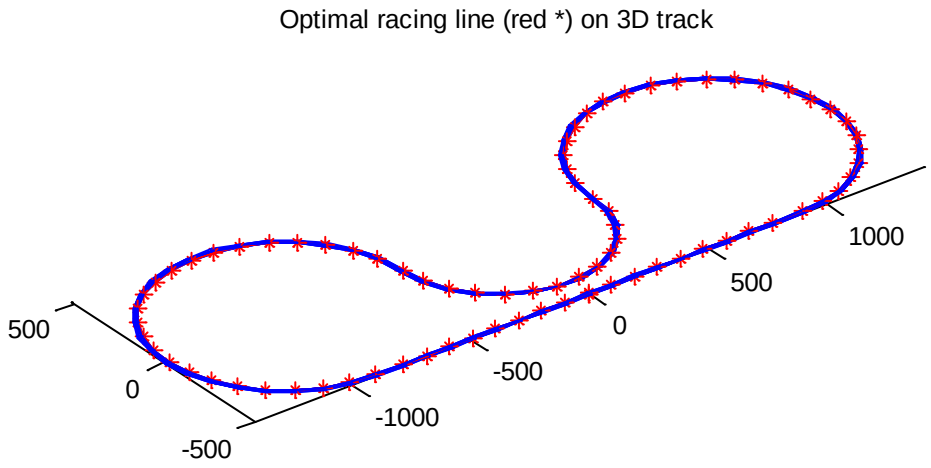


Figure 5.28. Optimal racing line on 3-D flower track

Displaying it in the 3-D display software, we can take a look from different angles and distances. Figure 5.29 is watching higher up from the south of the track.

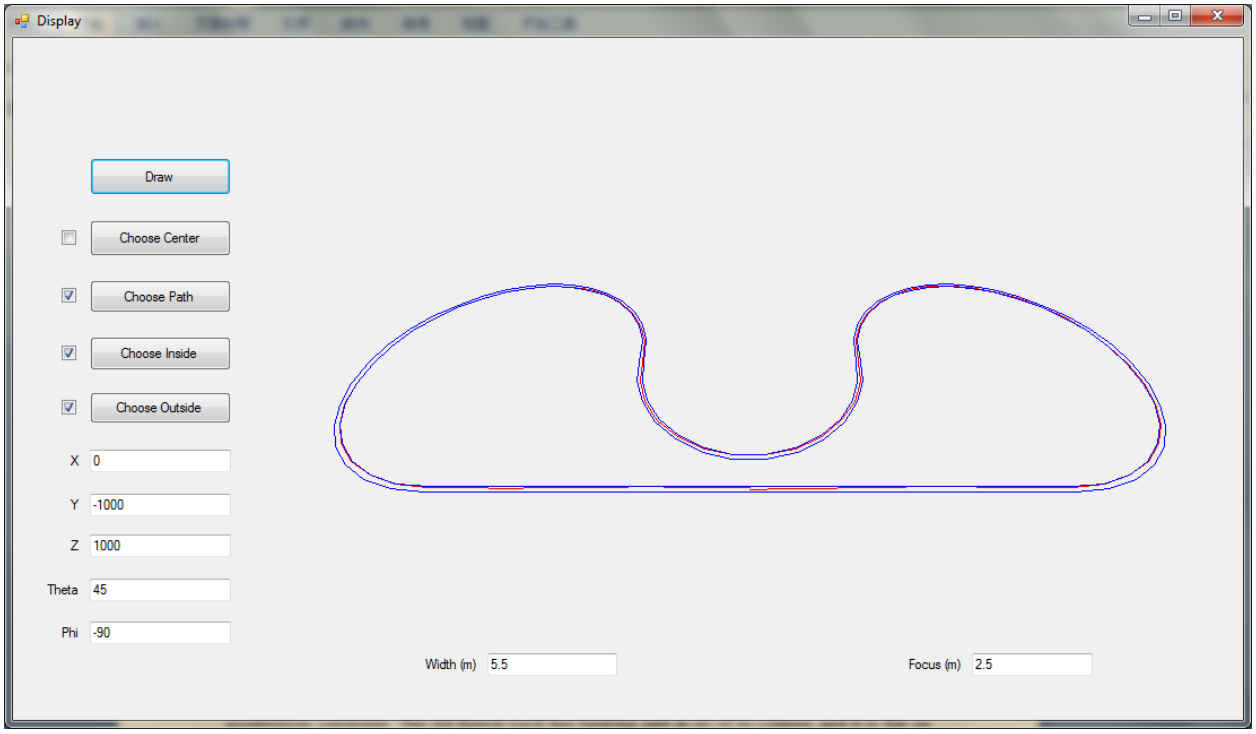
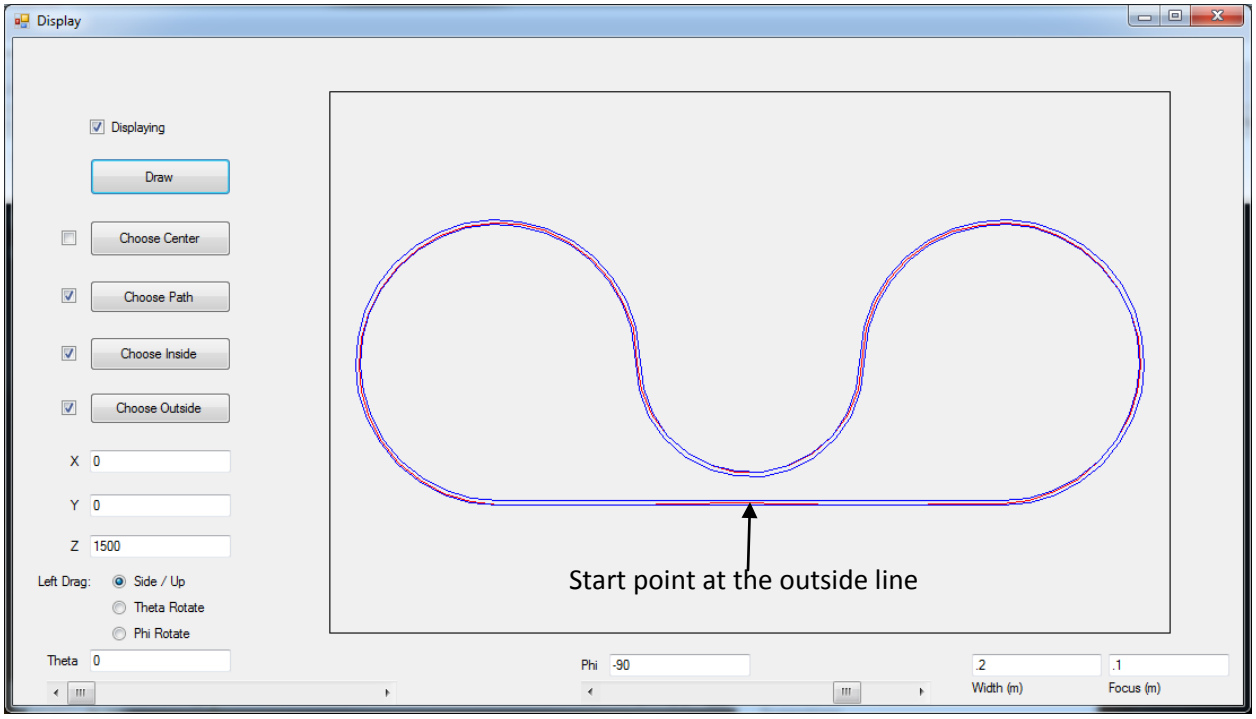
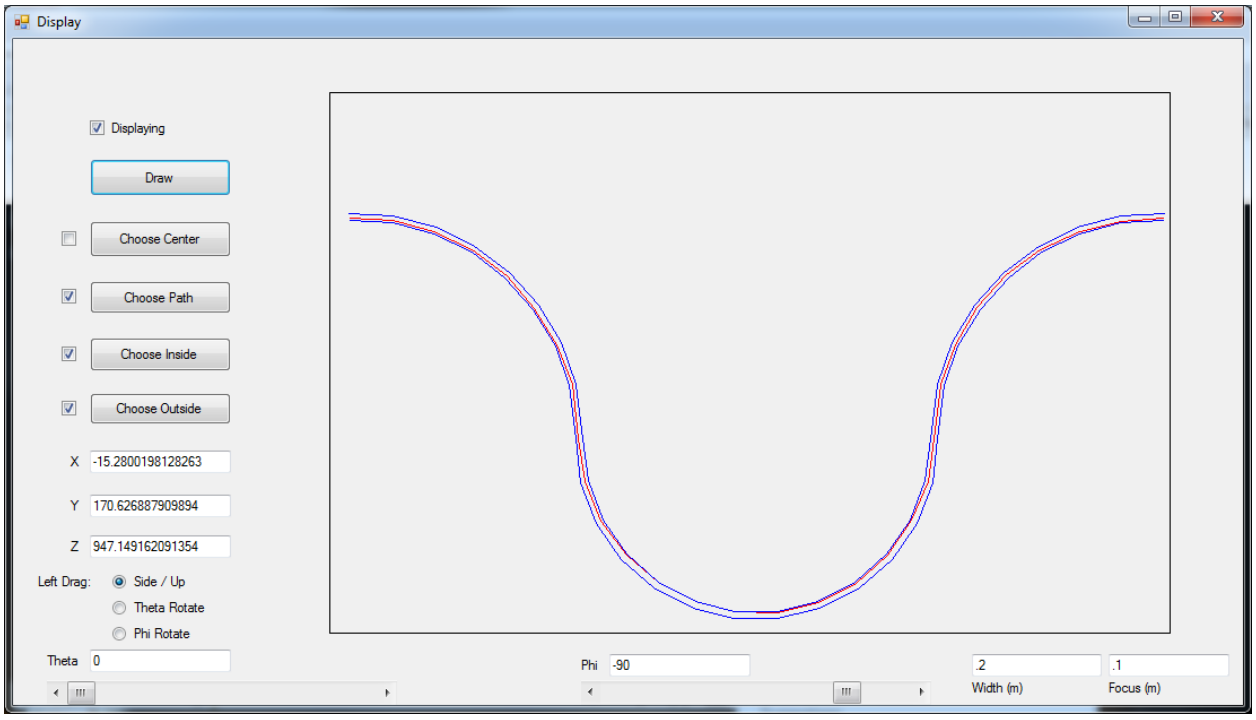


Figure 5.29. Three-dimensional display of optimal racing line in 3-D flower track with banking on all the corners

When the viewing position is in the center of the track and higher up, we can see a 2D projection view as in Figure 5.30.



(a)



(b)

Figure 5.30. The 2D projection of the 3D optimal racing line in the flower track. (a) shows the whole track and (b) shows a zoom in view of the three consecutive large cornering parts.

It is clear that the optimal racing line makes a smart solution. It follows a smooth route which touches several apexes. The Figure 5.30(b) shows clearly how the line goes along three consecutive corners. According to our calculation, it takes only 114.015 seconds to complete a lap. Because the corners are very large, the speed performance is also very good without much loss of speed during the turns. And the power constraint makes the positive acceleration part jump less. Let's look at the velocity graph in Figure 5.31.

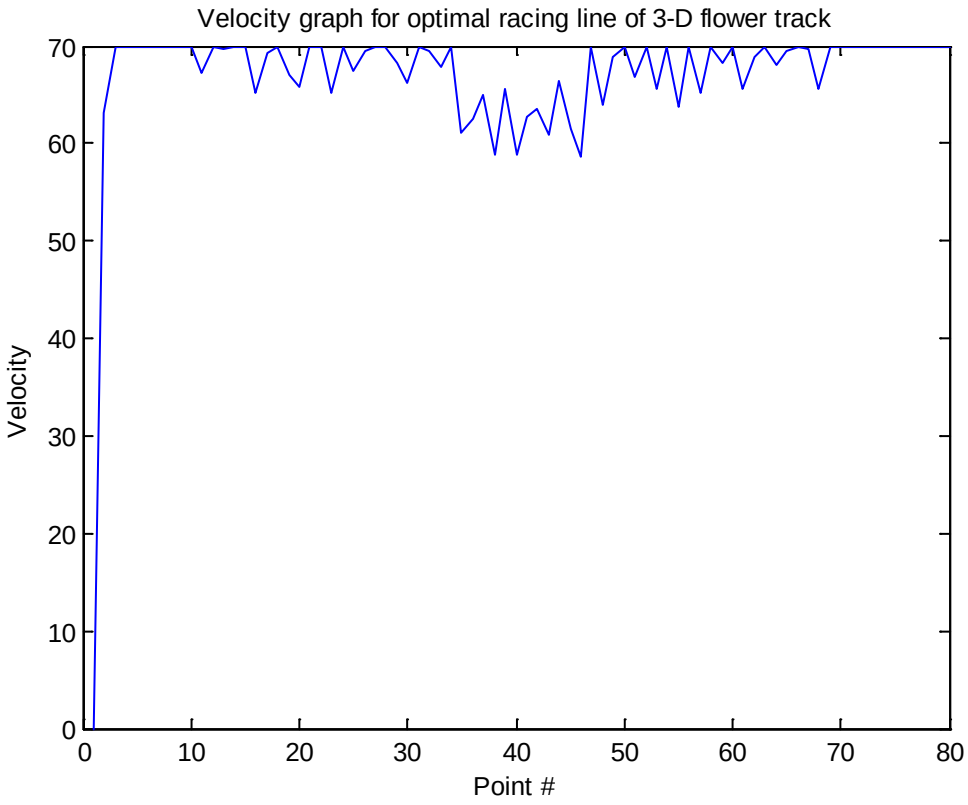


Figure 5.31. Velocity graph for optimal racing line of the 3D flower track

The figure shows that at many points the speed reaches the maximum allowed (70m/s), with the lowest speed at the smallest radius corner.

Now we are going to test the 3-D optimal racing line method for a rounded square track. The rounded square track is special in that it has four 90 degree corners and four segments of straight racing lines. We will test it in three scenarios: without banking or slopes, with banking but without slopes, and with banking and slopes. The results are shown in Figures 5.32, 5.33 and 5.34.

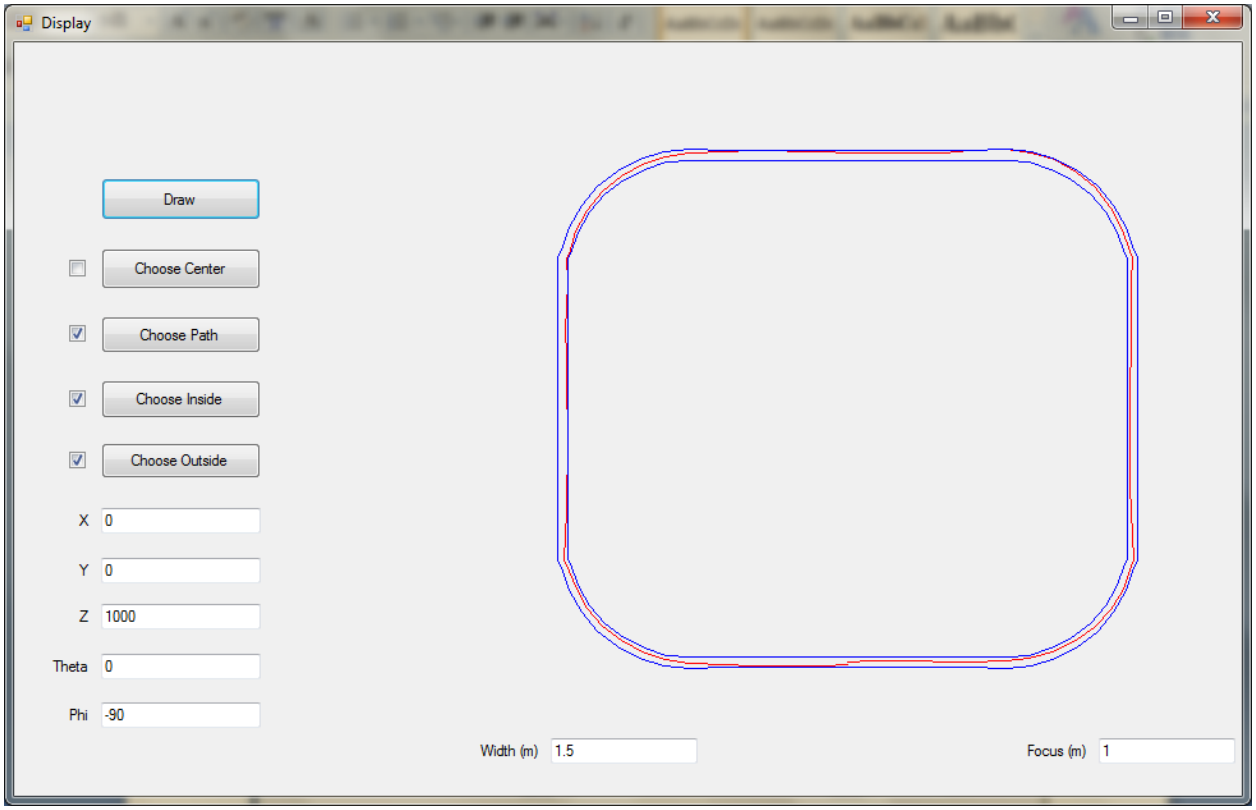


Figure 5.32. The optimal racing line generated for a rounded square track whose m=500, n=400 and r=80 and which does not include banked corners.

We can see that it does not necessarily have apexes at each corner. Whether or not an apex is needed depends on the road condition. Late apex ensures that the car has a large exit speed, but when the straight line part after the corner is not long enough, it may not be a good idea to take a late apex.

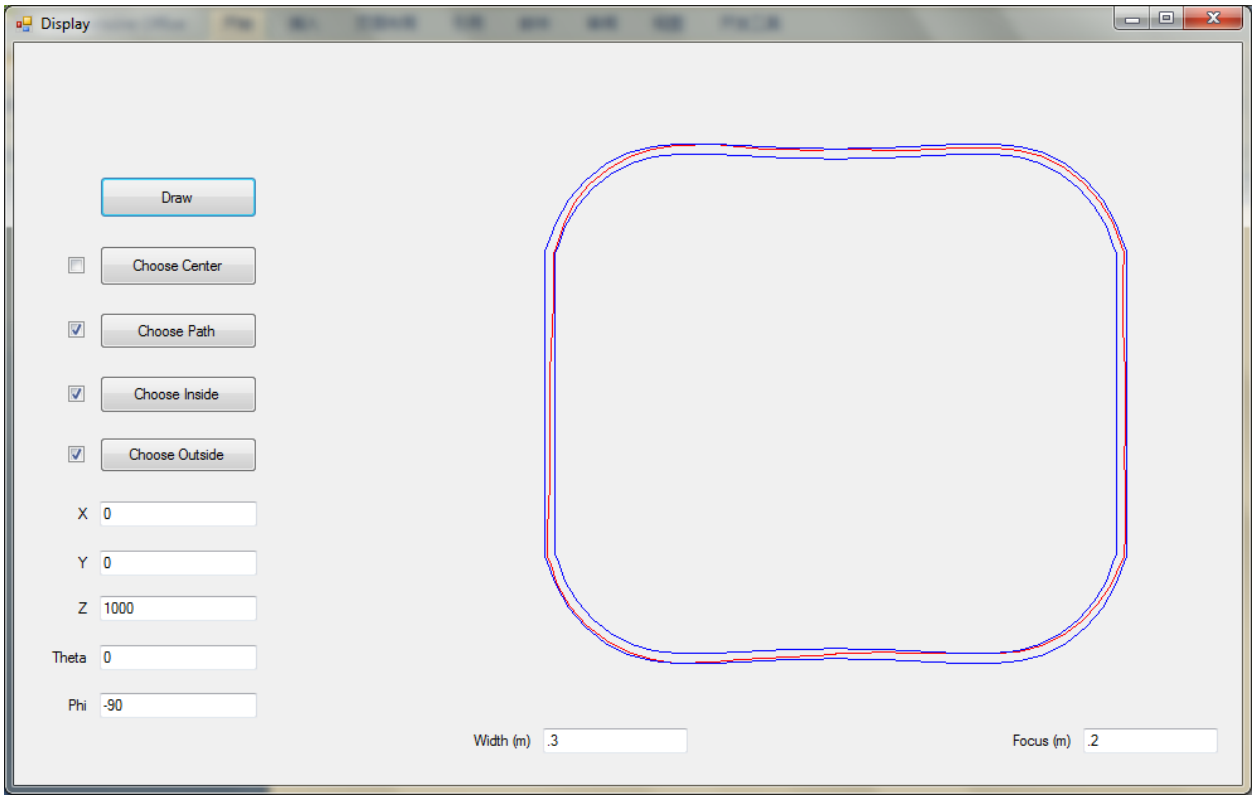


Figure 5.33. The optimal racing line generated for a rounded square track whose m=500, n=400 and r=80 and which includes banking.

The banking angle in Figure 5.33 is very small, so it does not have a very large effect on the racing line. We see two apexes and the other two corners do not have apexes.

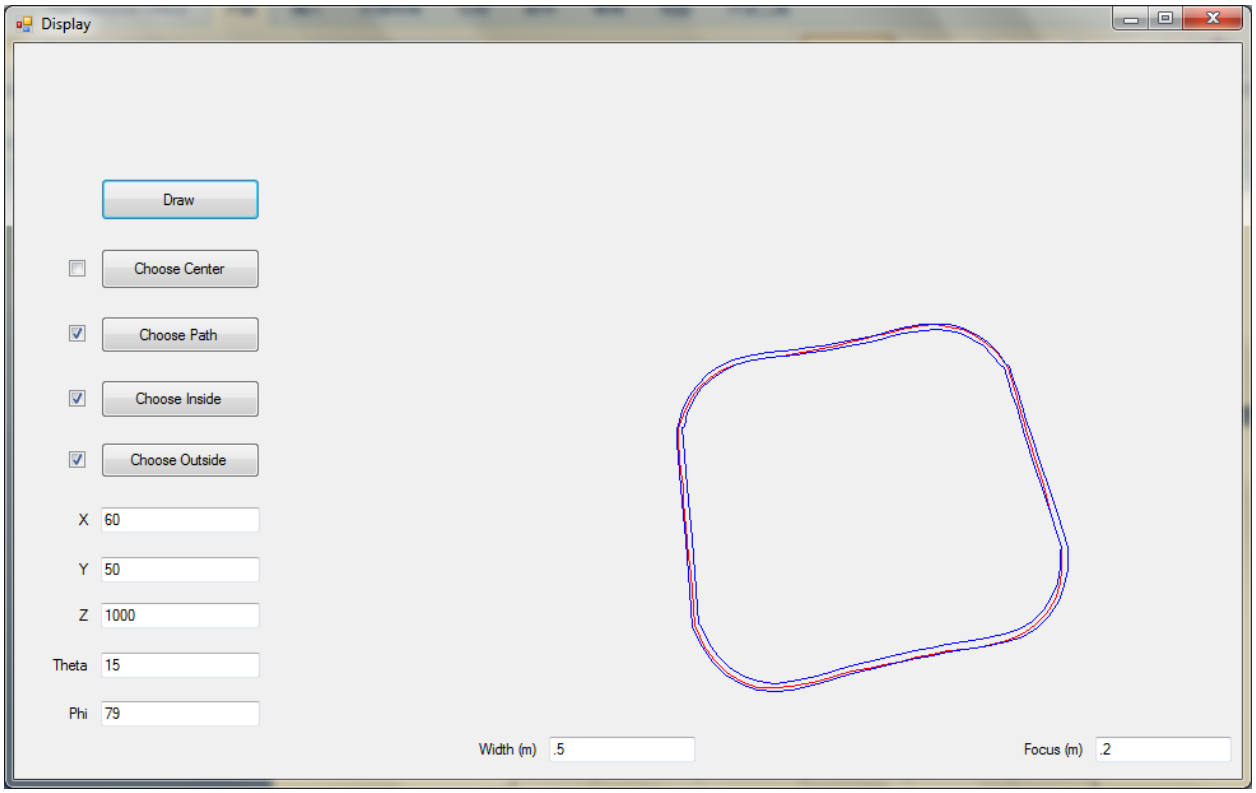


Figure 5.34. The optimal racing line generated for a rounded square track whose m=500, n=400 and r=80. This track contains both banked corners and up/down slopes.

With up and down slopes along the way, the optimal racing line may look a little less regular. But the essential point is still trying to maximize cornering performance to save time cost during the course of the race.

5.2.3 More result analysis

We may choose different numbers of points along the track picked for the same racing track with the same number of iterations. Here we are going to try 5 different numbers for the rounded square track. The rounded track has the following shape:

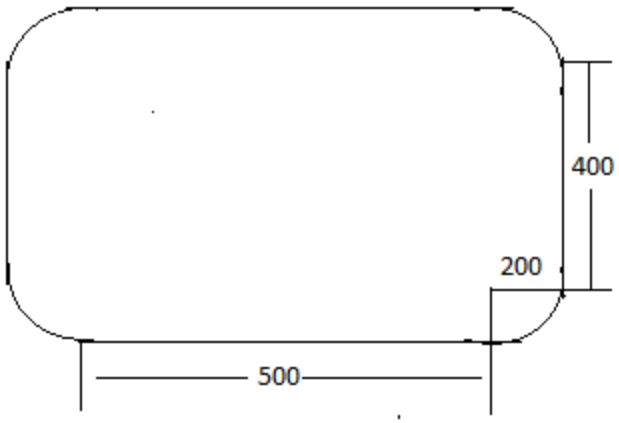


Figure 5.35. The rounded square shape track.
The table below shows the execution time and optimal time cost with different numbers of points along the track. Note that 80 points are picked along this track.

We are going to run 20,000 iterations for all of them. Different numbers of points are used to represent the same racing track. The optimal time cost result and the execution time will be different. Obviously the more points there are along the track, the more precise they represent the original track, and the more time it may cost to run the program.

Table 5.4. Relations of execution time, optimal time cost and number points along the track

	# of points along the track	Execution time (seconds)	Optimal time cost (seconds)
1	40	22	63.7886
2	60	38	62.8897*
3	80	48	67.9346
4	100	57	66.984*
5	120	72	71.5797

Note that for the two numbers with * in the time cost column, the time best racing line and the E best racing line generated are the same. For the other three, these two racing lines are not

the same, and the time cost to finish the track using time best racing line is less than that using the E best racing line.

Figures 5.36 and 5.37 analyze the relations between the execution time and the number of points along the track, and between the time cost of the optimal racing line generated and the number of points along the track.

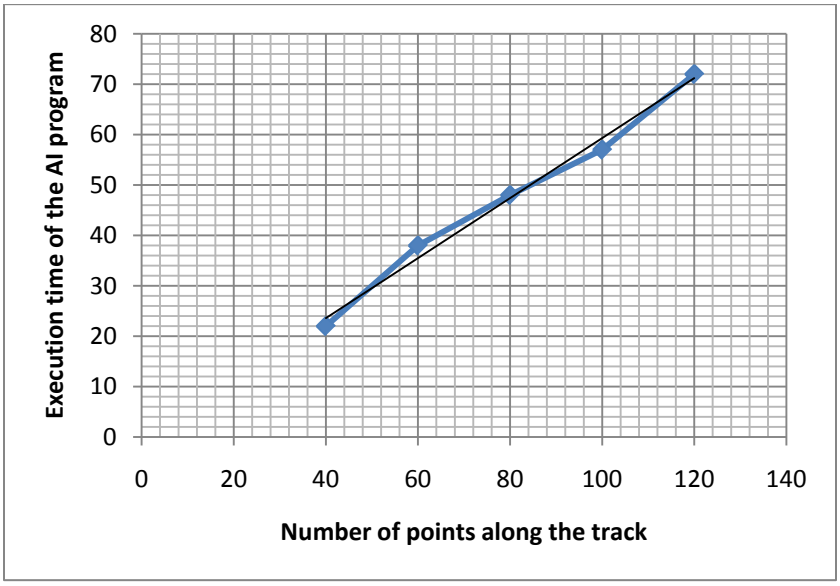


Figure 5.36. Execution time of the program vs. number of points along the track

From the graph above we can see that the program execution time almost changes linearly with the number of points along the track. The time complexity here is O(n). Note that we discussed before that the program complexity is O(mn) where m is the number of iterations and n is the number of points. Here we have a fixed number of 20,000 iterations, so the time complexity is O(n).

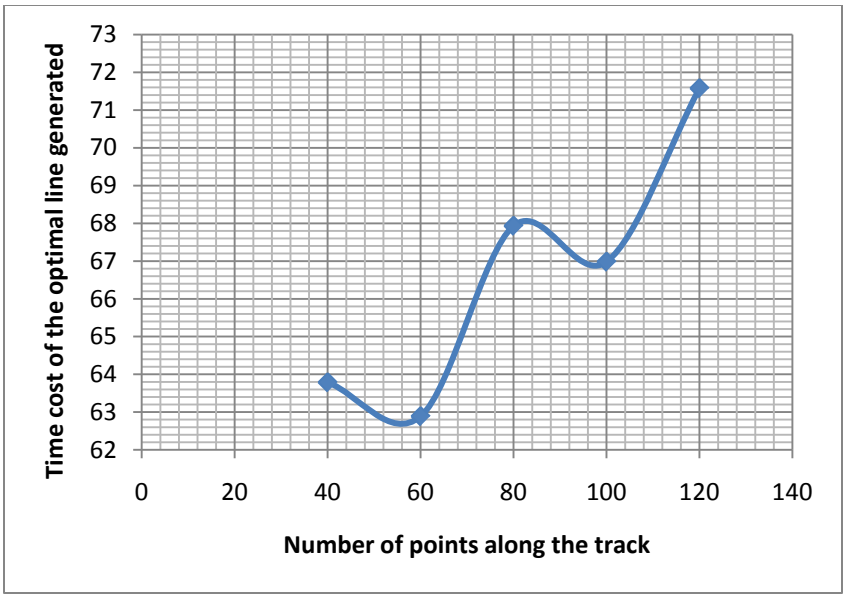


Figure 5.37. Time cost of the optimal racing line vs. the number of points along the track

On the other hand, we use different numbers of iterations for the same track with the same number of points picked along the track. The results are shown below (Table 5.5).

Table 5.5. Execution time, time cost result and number of iterations

	# of iterations	Execution time (seconds)	Optimal time cost (seconds)
1	2,000	5	60.2128
2	10,000	29	60.2880
3	20,000	59	59.6270
4	30,000	88	59.5317
5	50,000	146	59.1855
6	100,000	288	59.5565

Figures 5.38 and 5.39 analyze the relations between the execution time and the number of iterations, and between the time cost of the optimal racing line generated and the number of iterations.

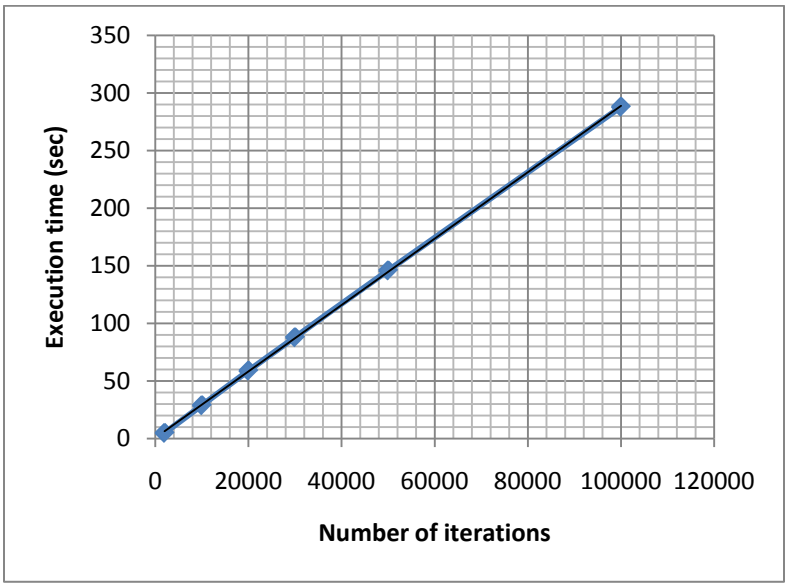


Figure 5.38. Execution time vs. number of iterations

From Figure 5.38, we can see that the execution time increases linearly with the number of iterations.

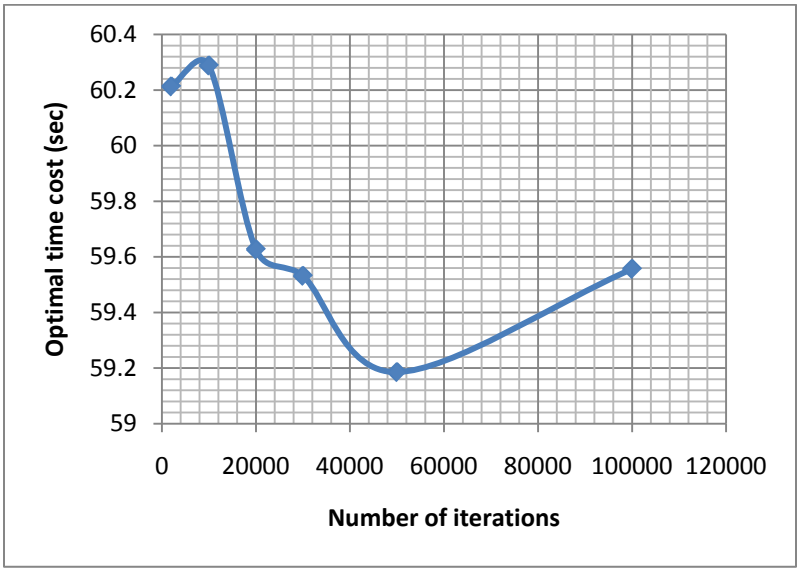


Figure 5.39. Optimal time cost vs. number of iterations

Figure 5.39 shows the relationship between the number of iterations and the optimization results. We can see that when the number of loops goes up, the time cost is generally going down. However, due to randomness involved in this method, a larger number of iterations does not always guarantee a better result. For example, 2,000 iterations is acting slightly better than 10,000 iterations, and 100,000 iterations is not performing better than 50,000 iterations in this case. However, generally speaking, better results are obtained with more iterations. We can see this by running with these specified loop numbers for many times and taking the average values.

Besides, the big-small step strategy is helpful in covering wider possibilities of paths and obtaining better results. However, the number of small steps in a big step can be a factor in its effectiveness. In the table below we study the different numbers of small steps in one big step for the same racing track. Note that on the 1st row, the step size of big step is just the step size of small step, i.e., no big-small step strategy is used. Here for row 1, the data is the result after 90,000 iterations, while for the others they all run for 30,000 iterations. The reason is that in the big-small step algorithm, firstly an optimal racing line considering only big steps is generated after 30,000 iterations, and this is followed by filling in the small steps for the time best and E best racing lines consisting of big steps, each with 30,000 iterations. So in total it is comparable to 90,000 iterations of the usual method. The results are listed in Table 5.6.

Table 5.6. Relations of execution time, optimal time cost and number of small steps in a big step

	Number of small steps in one big step	Execution time (sec)	Optimal time cost (sec)
1	1	247	59.4582
2	2	293	56.6146
3	3	273	56.6531
4	4	270	56.6357

Figures 5.40 and 5.41 show the influence of number of small steps in one big step, i.e., the step size of big step divided by the step size of small step.

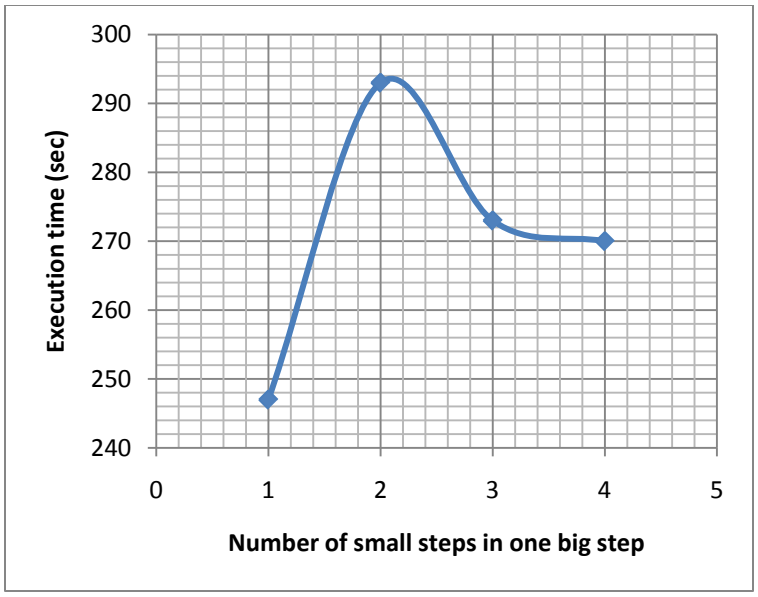


Figure 5.40. Execution time vs. Number of small steps in one big step

We can see that for 2, 3 and 4, the execution time of the program is decreasing with the increase in number of small steps in one big step. This is because the bigger number of small steps in one big step, the fewer the big steps, so it can run faster and faster in the first phase of this algorithm. But there is also a limit for this. To take an extreme example, when the size of a big step equals the total length of the track, it is equivalent to having no big-small step strategy at all.

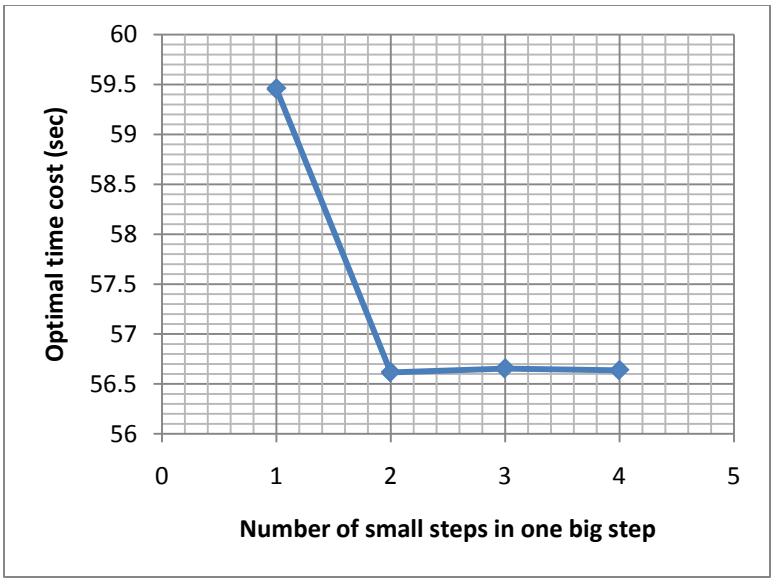


Figure 5.41. Optimal time cost vs. number of small steps in one big step

The big-small step algorithm is clearly very useful. Even when using 2 small steps for each big step, the improvement is impressive: about 6% smaller time cost.

6. Integrated approach

6.1 Introduction to the integrated approach

Racing line optimization is complicated in that there is not a method that is best for all situations. Which method to choose will depend on the features of the track. The good news is that the artificial intelligence method is universally applicable to any kind of 2-D or 3-D tracks with any slopes or banking. However, it is comparatively slow and not always the best way of finding the solution. In some specific cases, such as a track with regular large corners connected by long straight lines or curvy lines, the Euler spiral method is definitely better and faster for the corner part, while the artificial intelligence method can be used to deal with the rest of the track. What's more, if the curvy line is long and curvy in a regular way, we can use a nonlinear solver to solve for the points. As a result, it is important to be able to flexibly integrate different methods at different parts of the track. Some implementation results of the integrated method will be discussed in Chapter 7.

6.2 Implementation and results

The integrated approach starts with analyzing the shape and special features of the racing track. Basically, we take advantage of the fast speed and precise results of Euler spiral method when there are corners, and connect the other parts by either the AI method or the nonlinear solver method depending on the problem size. After that we connect the different segments of racing line along the track and calculate the total result.

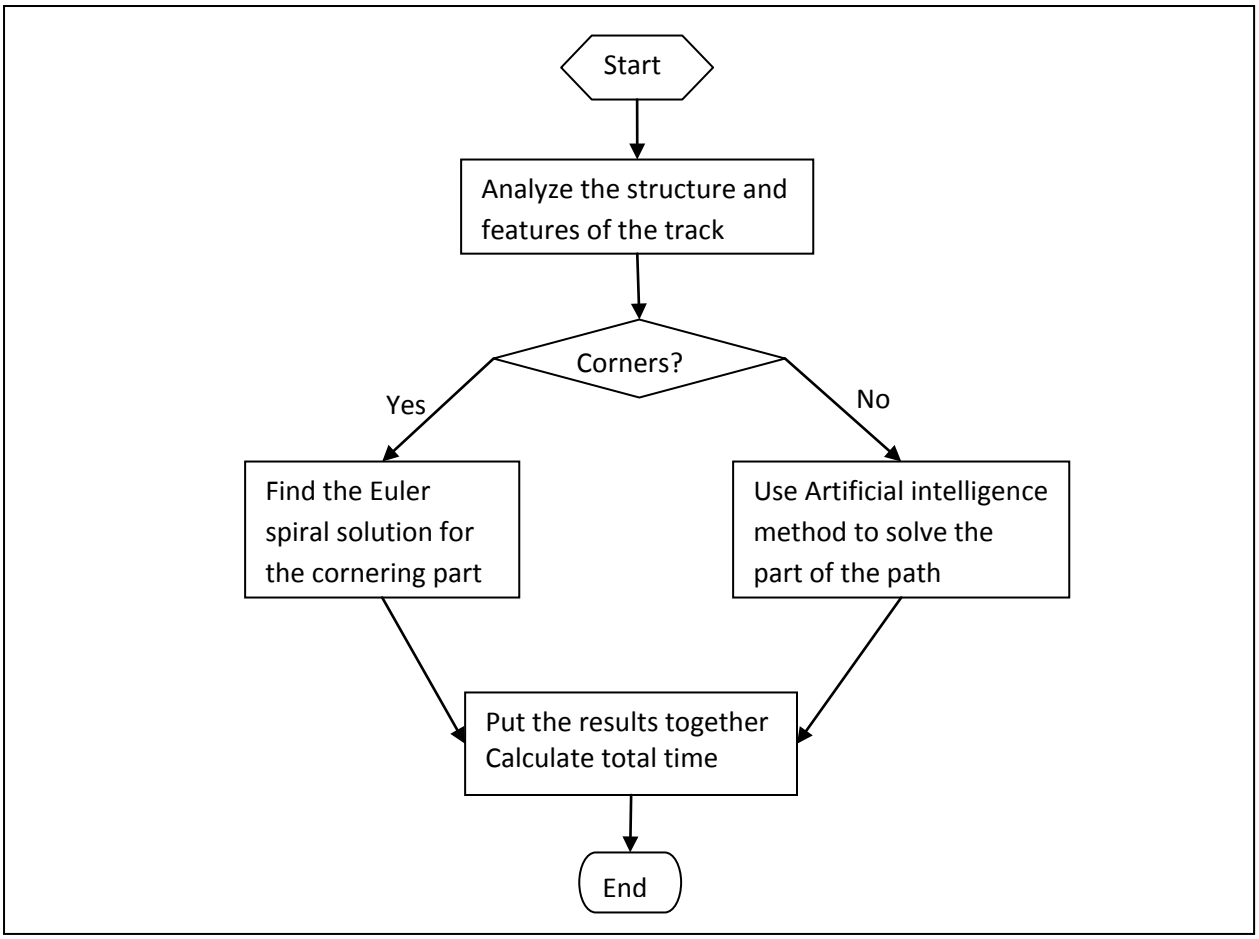


Figure 6.1. The flow chart for the integrated method to find the optimal racing line

Assume that there is a rounded square track with the following parameters $m = 500,\ n = 400,\ r = 20 + 16/2 = 28$.

Some results below show the 90 degree turn of a large racing track whose width is 16 meters and radius is 200 meters. The red line represents the Euler spiral optimal racing line, while the blue line represents the inside edge of the racing track.

(1) When the end of line radius $R_s = 100$, the result is shown in Figure 6.2.

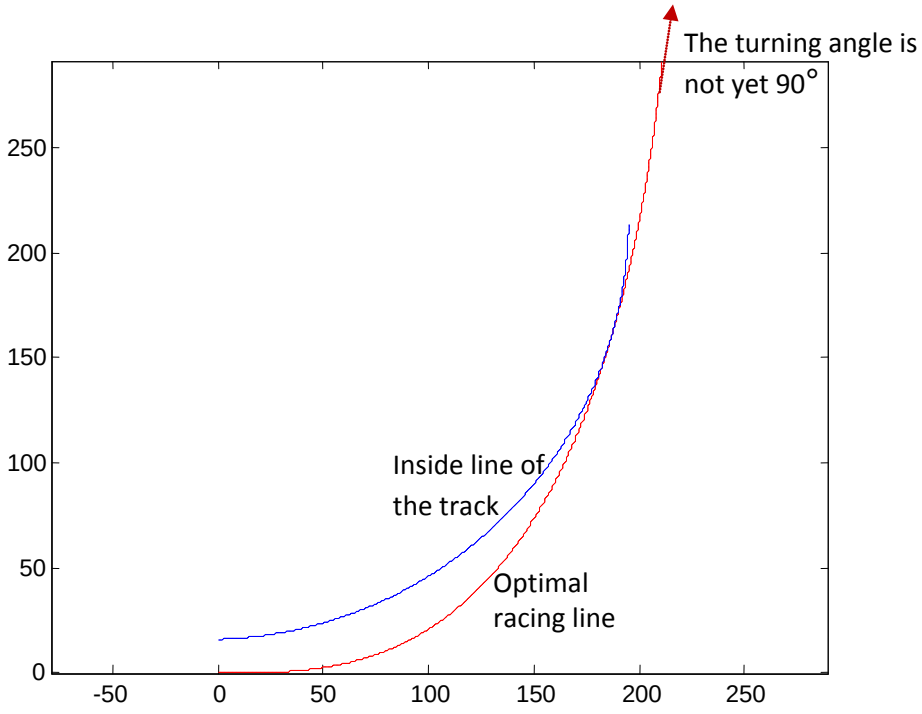


Figure 6.2. MATLAB plot of the Euler spiral shape optimal racing line hitting one apex on the inside line of the 90-degree corner track when the ending radius R_s=100

The time cost for making a 90 degree turn is $t = 8.52548$ seconds. Obviously at this point the red line did not bend 90 degrees yet. This means the designated R_s is too large. We can try some smaller values of R_s.

(2) When the end of line radius $R_s = 50$, the result is shown in Figure 6.3.

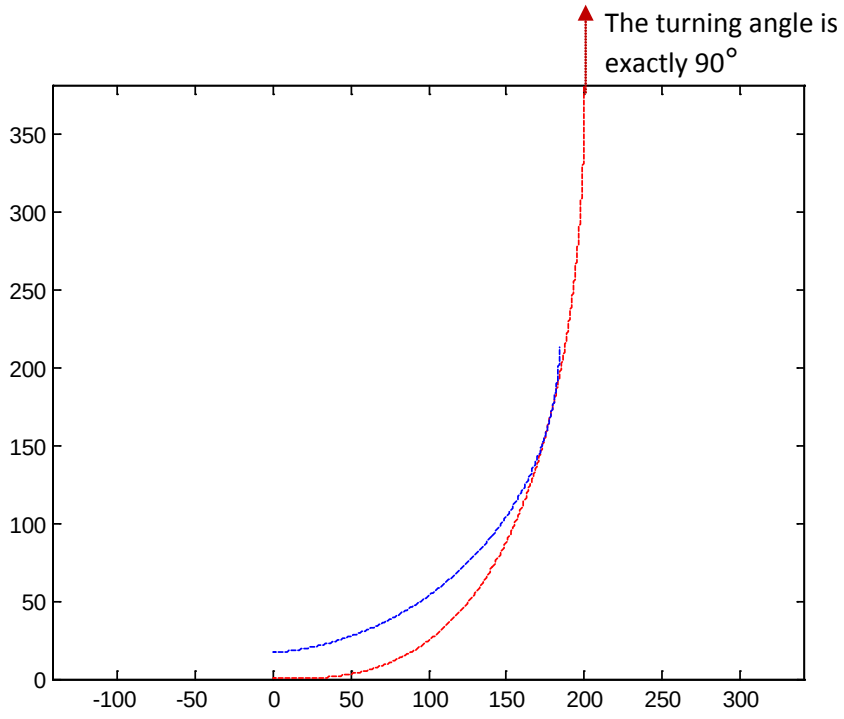


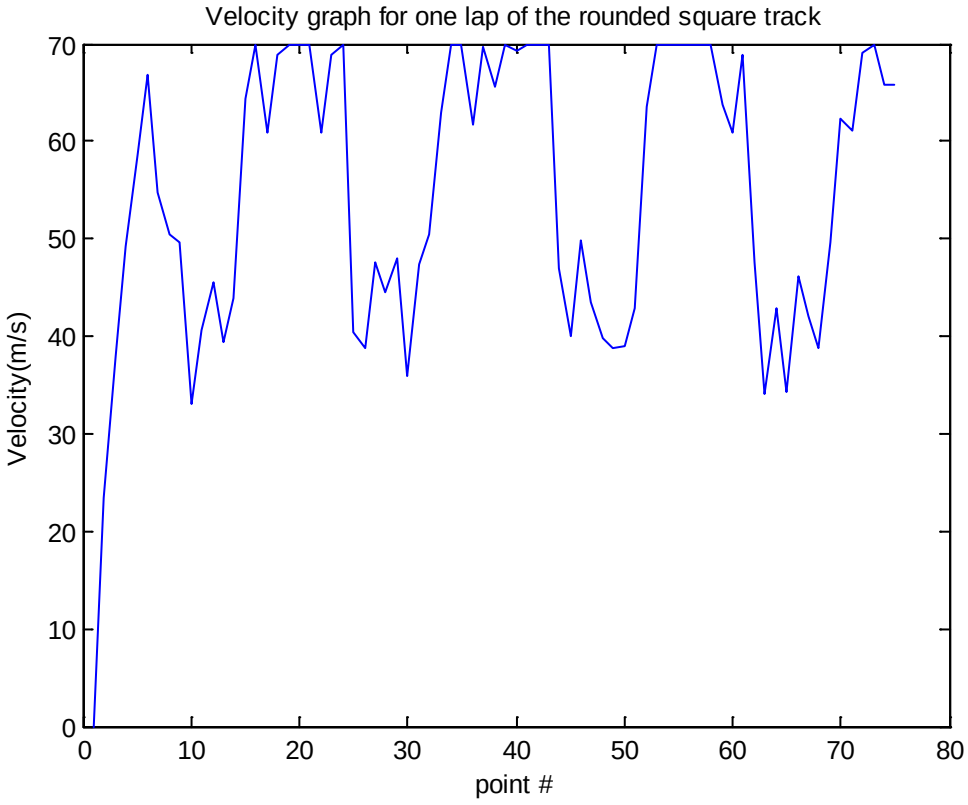
Figure 6.3. MATLAB plot of the Euler spiral shape optimal racing line hitting one apex on the inside line of the 90-degree corner track when the ending radius R_s=50

The total time cost is $t = 9.70311$ seconds. For the red line, the two points at the beginning and the end are $(0,0)$ and $(200,\ 374.6)$.

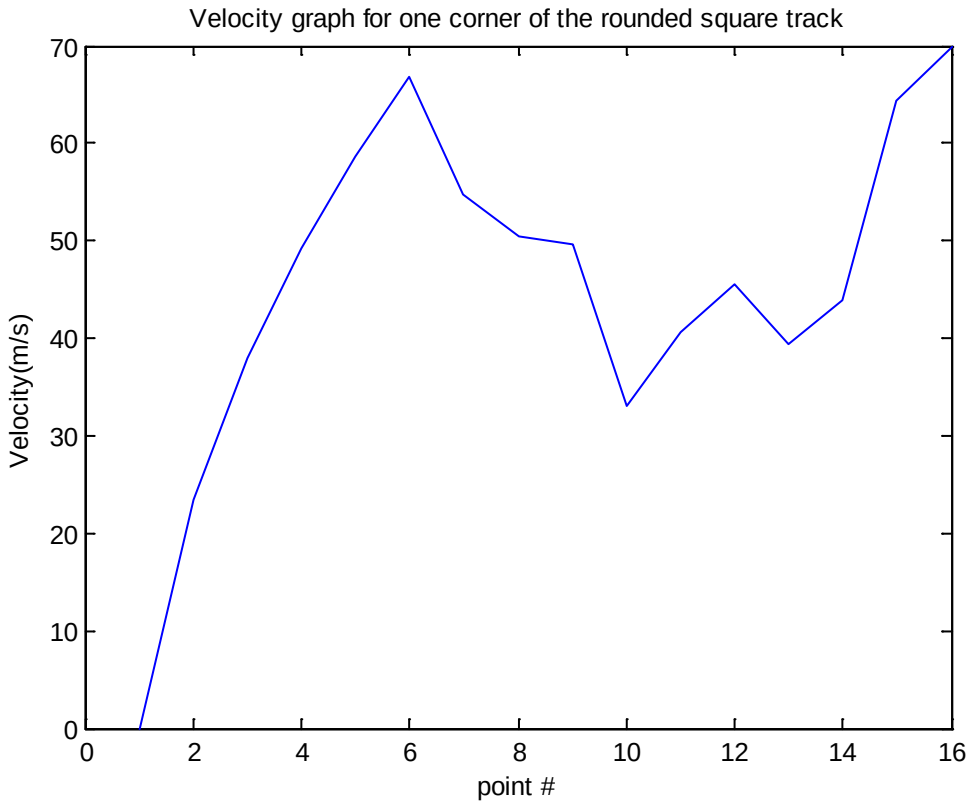
According to the symmetry of the rounded square racing track, $(m+2r)$ should be at least 200+374.6=574.6 to let all four corners have an Euler spiral part of the racing line. For the rounded square track of $m = 500,\ n = 400,\ r = 208,\ width = 16$, the solution by purely artificial intelligence method is in Figure 6.4.

Figure 6.4. The artificial intelligence method results for a rounded square racing track

The velocity graph for the optimal racing line in Figure 6.4 is as follows (Figure 6.5):


Velocity graph for one lap of the rounded square track
Velocity(m/s)
point #
0
10
20
30
40
50
60
70
80

(a)



(b)

Figure 6.5. Velocity graph of the optimal racing line for a rounded square track with m=500, n=400, r=208, generated by the Artificial intelligence method. (a) shows the velocity for a whole lap; (b) shows the velocity for a single corner.

From Figure 6.5 we can see that for a single corner, the speed goes down and then up again. For the whole lap, the velocity graph tends to be bouncing and it is clear to see that there are four corners along the track. Note that in order to make the execution time shorter, points are picked with a relatively large distance between each other. That's why the graph does not look very smooth. We should take it as showing that the speed needs to change from v(i) to v(i+1) in a smooth and reasonable way from point i to point i+1.

The total time cost of the optimal racing line generated by the AI method is 59.022 seconds. However, using the integrated method, the four corners use the Euler spiral in Figure 6.3, and connect with the straight parts using the AI method.

$$
\begin{aligned}
t &= t_{corner} + t_{other} \\
&= 9.70311 \times 4 + t_{AI} \\
&= 54.5496 \text{ seconds}
\end{aligned}
$$

This result is better than the 59.022 seconds needed by the pure AI method, and it also runs faster. The velocity graph for the corner is shown in Figure 6.6.

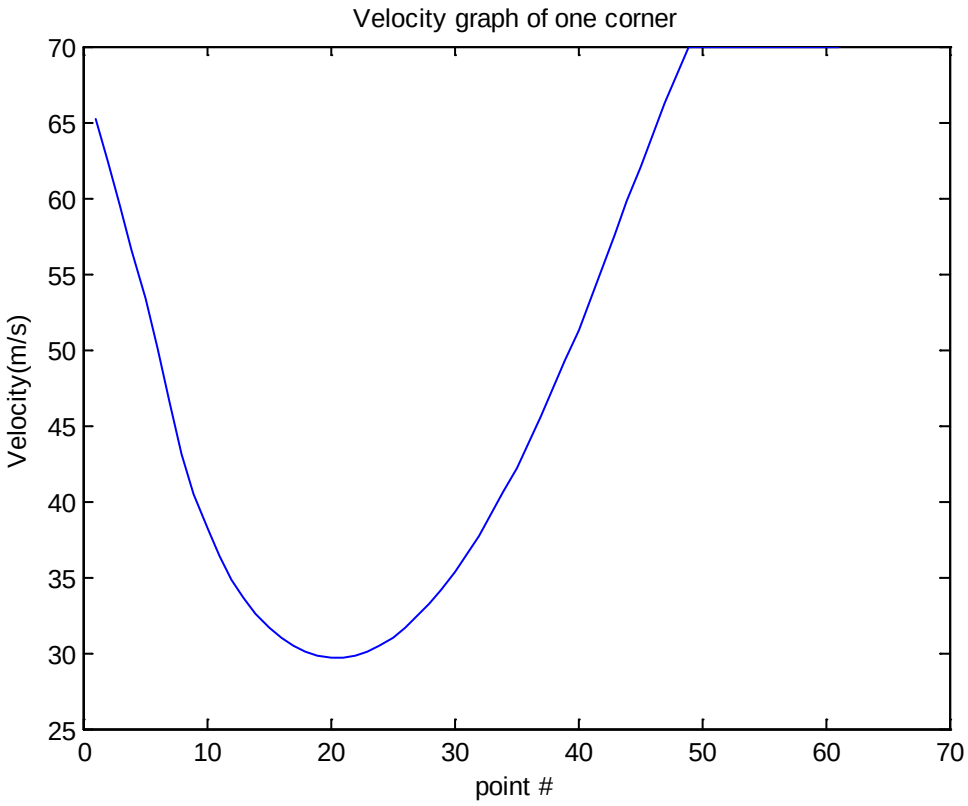


Figure 6.6. The velocity graph of the corner part using the Euler spiral method

From the velocity graph above we can see that the exit speed will reach the maximum value allowed 70 m/s. Moreover, it already got to 70 m/s before it completes the turn. The speed along the line is very smoothly changing and there are no sudden jumps.

We tested the integrated results with different R_s values from 20 to 1000 with the incremental step of 20. Some interesting results are shown in Figure 6.7. The x coordinate grows larger when R_s increases, while the y coordinate grows smaller when R_s increases. The time cost sharply decreases when R_s increases. Then it reaches a flat non-growth phase,

and starts decreasing again until it reaches 3 seconds. The reason is that when R_s is very large, the line between the start point and the end point will be very close to a straight line, and according to the *straight line rule* ('for a plane surface, the shortest distance between two points is a straight line'), the time cost will gradually reduce to 3 seconds which is the time cost to go through the straight line connecting the start point and the end point.

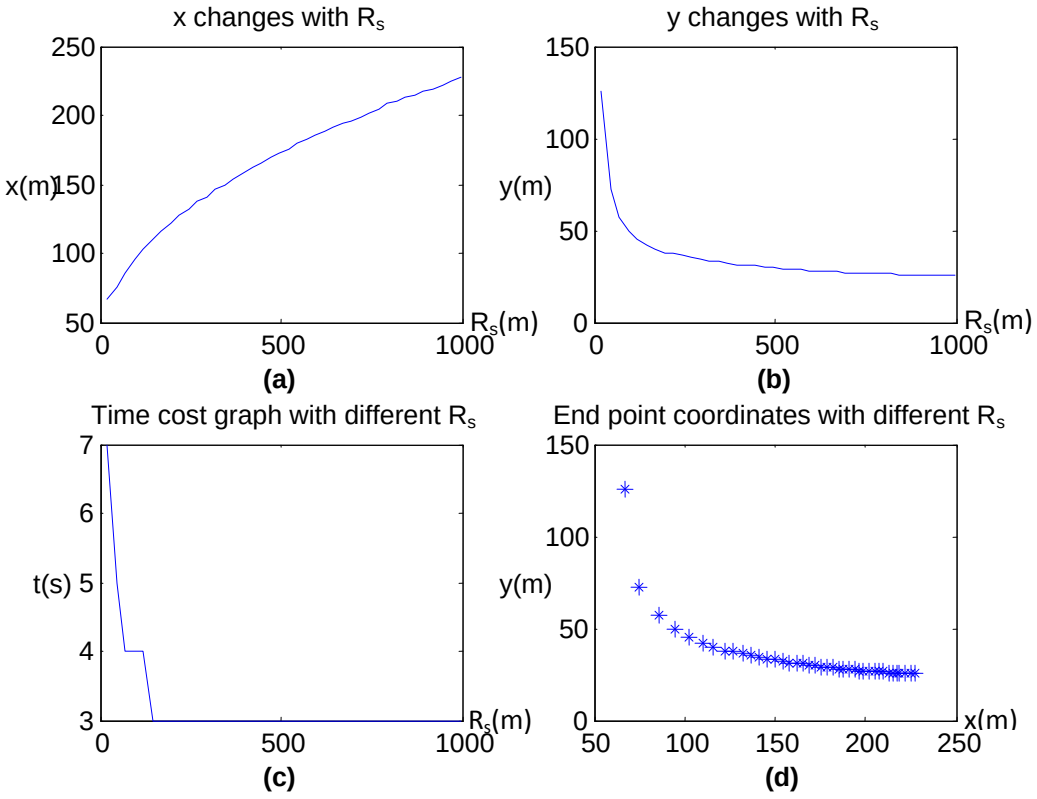


Figure 6.7. Test results for different values of R_s. (a) shows how x changes with R_s. (b) shows how y changes with R_s. (c) displays the different time costs when we use different R_s values. (d) shows the positions of the end points with different R_s values.

From the end point coordinates graph we can see that there are some constraints for the proportions of the track to make the Euler spiral cornering work well. If the straight line is not long enough, it will not give enough space for the Euler spiral to spread out and make a complete turn of 90 degrees. And even if the 1st corner part is completed by an Euler spiral, there may be not enough room to prepare for the 2nd corner.

Actually, it will not be always optimal to use an Euler spiral in every corner. According to the output, even when it is applicable to use Euler spirals for all 4 corners, it may **not** be good if the straight part is comparatively too small or the radius of the corner is comparatively too large.

When $m = 500,\ n = 200,\ r = 50,\ N = 80$, the time cost by purely AI is 46.5598 seconds. With integrated method, end point (92.0879, 150.6400). The total time cost is 43.2257 seconds. The integrated method is performing slightly better.

When $m = 500,\ n = 200,\ r = 30,\ N = 80$, the time cost by purely AI is 41.1489 seconds. Use integrated method, and the end point is (75.4566, 141.9656). The total time cost is 37.8588 seconds. The integrated method is performing slightly better.

When $m = 500,\ n = 200,\ r = 30,\ N = 320$, the time cost by purely AI is 41.1489 seconds, the same as before. But using integrated method, we know that end point (75.4566, 141.9656), and the total time cost is 35.8588 seconds. The integrated method is performing even better.

The advantage of the integrated method will be more obvious when the straight part takes up a significant percentage of the track, and when more points are included to describe the track.

One key point of change in the original artificial intelligence method is that there are more inputs. For the original methods, the only input is the track information (Figure 6.8).

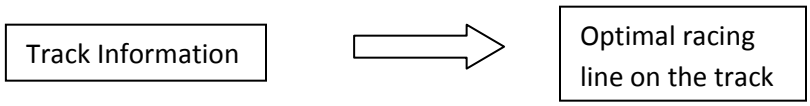


Figure 6.8. Input/output relations for the three methods that we mentioned above: Euler spiral method, nonlinear programming solver method and Artificial intelligence method.

Now there are four more inputs: starting position, finishing position, starting speed, finishing speed (Figure 6.9). These inputs make it possible to connect together results from different methods for different parts of the track. For example, the end speed of one segment equals the start speed of the next segment.

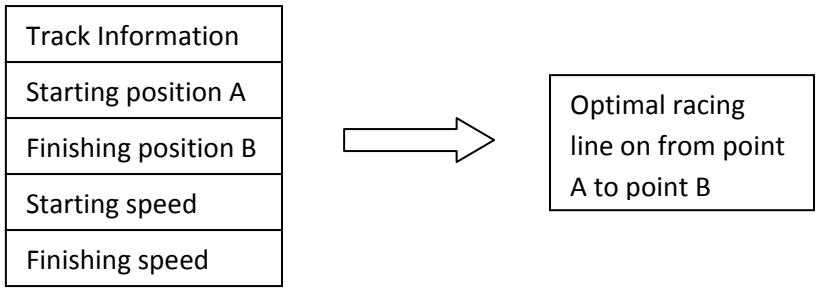


Figure 6.9. Input/output relations for the integrated method

7. Summary

In this thesis we investigate the problem of optimal racing line selection based on racing track and car information. Four methods introduced in the previous four chapters all work out in some situations. The artificial intelligence method is widely applicable, and the integrated method absorbs the advantages of the artificial intelligence method, but speeds up the process by making decisions about which method to use on different segments of a whole racing track.

7.1 Comparison of different methods

Each method introduced above has its own advantages and disadvantages. The comparison is made in detail in Table 7.1.

Table 7.1. Advantages and disadvantages of the four different methods that are introduced above

	Advantage	Disadvantage
Euler spiral method	a. Guarantee of near-optimum in cornering part b. Natural achievement of an apex, and late apex can assure high exiting speed c. Velocity is changing continuously d. High computing speed for the appropriate type of track	a. Not flexibly applicable to irregular shapes of racing tracks. b. The model is not guaranteed to be the best when more factors other than the lateral friction are considered c. Not applicable for complex 3D situations
Nonlinear solver method	a. Very fast execution compared to artificial intelligence method using the specific nonlinear solver program b. Comparatively easy to write and modify.	a. Many approximations have to be made to make the program formulation recognizable by AMPL b. All different kinds of nonlinear solvers have their own features and they cannot guarantee not to become stuck in a local optimum

Artificial intelligence method	a. Very flexible to find the optimal racing line for all kinds of irregular and complicated 2-D or 3-D racing tracks b. Guaranteed to get a racing line that can beat most other racing lines and an acceptably good time cost.	a. The velocity graph will not be automatically smooth and many adjustments are needed to make the velocity changes practical with the limitation of accelerations. b. The execution time may be long when the track is very long
Integrated method	a. A good combination of the mathematically and physically optimal Euler spiral curve and the optimal line found by the Artificial Intelligence method. b. Usually will be an improvement to the pure AI method c. More widely applicable to different shapes of tracks	a. Not applicable for all shapes of tracks. Pre-analysis of the track shape is required b. No guaranteed improvement of the pure AI method because sometimes the shape does not apply to the Euler spiral method very well, or the radius isn't small enough to make the tradeoff between shorter distance and larger turning speed.

7.2 Usage of results from our research

The methods described in this thesis can be used to calculate the optimal racing line for any 2-D or 3-D racing track. When the actual racing line is too long, the integrated method can be used to separate the original racing track into several segments. Given more iterations and longer time, the artificial intelligence method can obtain better results with lower time costs. There are parameters for different car features and road features, so different values can be substituted in for different cases.

7.3 Conclusions and future work

This thesis studies the optimal racing line for a given track and type of car. We find some common features about optimal racing lines through the different methods that we describe in this paper. The optimal racing lines usually make use of both the corner parts and the straight parts of a racing track. There are often apexes reached around corners. On straight lines a car can mostly go in one direction and keep on speeding up to its upper limit, but at the same time it should make adjustments early enough for a good cornering strategy.

As cornering performance greatly determines the outcome of a racing match, a good strategy to determine what racing lines to take and what speeds to use at non-straight parts is critical for any racer. The optimal cornering strategy depends on many factors, such as how much straight line there is before and after the corner, whether the corner is banked or not, how much the corner is banked and whether there are any slopes on the road. Generally, if the straight line after the corner is long enough, a late apex will be preferred to a middle apex. A middle apex is more neutral but most of the time not the best. Also when there is more banking, the car can go through a line that has smaller turning radii.

We also find that for 2-D tracks the racing lines with smallest time cost and with smallest value of $E = \int k^n ds$ (for $n = 0.2,\ 0.5,\ 1$ and 2) are very similar but not the same. This is why the Euler spiral can be used to approximate optimal cornering in a convenient way since it comes close to a minimal value of $\int k^2 ds$. For general 3-D tracks, minimizing a modified definition of E with $n = 0.5$ can achieve a result close to the optimal racing line.

For a specific problem, the four methods in this thesis all have their own pros and cons, so we should choose an appropriate method for each track. As an example, the integrated method can be used for a rounded square track because it has four complete 90 degree turns connected by four straight parts.

There is some more work to be done in the future on optimal racing line selection. Looking for a mathematical model for optimal cornering in 3-D conditions similar to the Euler spiral in 2-D is an interesting topic. Different models of cars and different kinds of racing tracks need attention. The present work can be used as a starting point for future investigations.

Bibliography

1. Kerry Spackman and Sze Tan, When the turning gets tough, New Scientist magazine, vol 137 issue 1864, March 13, 1993, pages 28-31.
2. Wikipedia, Formula One car, http://en.wikipedia.org/wiki/Formula_One_car, as of April 25, 2010, 21:38 GMT.
3. Wikipedia, Downforce, http://en.wikipedia.org/wiki/Downforce, as of April 25, 2010, 21:42 GMT.
4. Wikipedia, Ground effect in cars, http://en.wikipedia.org/wiki/Ground_effect_in_cars (as of July 25, 2010, 21:43 GMT).
5. B. K. P. Horn, The Curve of Least Energy, ACM Transactions on Mathematical Software, vol 9, issue 4, Dec. 1983, pages 441-460.
6. Benjamin B. Kimia, Ilana Frankel and Ana-Maria Popescu, Euler spiral for Shape Completion, International Journal of Computer Vision, vol 54, numbers 1-3, Aug. 2003, pages 159-182.
7. Thomas Gustafsson, Computing The Ideal Racing Line Using Optimal Control, Department of Electrical Engineering, Linkopings University, 2008.
8. Johan Åkesson, Tove Bergdahl, Magnus Gäfvert, and Hubertus Tummescheit. Modeling and Optimization with Modelica and Optimica Using the JModelica.org Open Source Platform, Modelica Association, Sep. 2009.
9. Michael E. Tipping, Mark Andrew Hatton, Ralf Herbrich, Racing Line Optimization. United States patent US 2007/0156327, Jul. 5, 2007.
10. Floris Beltman, Optimization of ideal racing line, BMI Paper, 2008.
11. H Makino, CIRP Annals, Manufacturing Technology, vol 37, issue 1, 1988, pages 25-28.
12. Wikipedia, Automobile drag coefficient, http://en.wikipedia.org/wiki/Automobile_drag_ coefficient, April 25, 2010, 22:30 GMT.
13. AW Nutbourne, PM McLellan, RML Kensit - Computer-Aided Design, vol 4, issue 4, July 1972, pages 176-184.
14. Alessandro Tasora, Real-Time Simulation of a Racing Car, 9th International Workshop on Research and Education in Mechatronics, Bergamo, Italy, September 18-19, 2008.

15. J. Åkesson. Optimica-An extension of Modelica supporting dynamic optimization. 6th International Modelica Conference, pages 57–66, Bielefeld, Germany, Mar. 3-4 2008.
16. R Baudille, ME Biancolini, L Reccia, An Integrated Tool For Competition Go-Kart Track Analysis, The 30th FISITA World Congress 2004, 2004.
17. D. M. Gay， Hooking your solver to AMPL, Technical Report 97-4-06, Computing Sciences Research Center, Bell Laboratories, 1997.
18. Z. Yao and A. Joneja, Path generation for high speed machining using spiral curves, Computer-Aided Design & Applications, 4:191–198, 2007.
19. Henry A. Watts, Secrets of Solo Racing, Expert Techniques For Autocross & Time Trials, Loki Publishing Company, 1989, ISBN 0-9620573-1-2.
20. Carroll Smith, Drive to Win, The essential guide to racecar driving, Carroll Smith Consulting Inc, 1996, ISBN 0-9651600-0-9.
21. Yoshiaki Kawajir, Carl Laird, Andreas Waechter, Introduction to IPOPT: A tutorial for downloading, installing, and using IPOPT, Carnegie Mellon University June 29, 2010
22. R. J. Vanderbei, LOQO: An interior point code for quadratic programming, Optimization Methods and Software, vol 11/12, part 1/4, pages 451-484, 1999.
23. Philip E. Gill, Walter Murray, and Michael Saunders, SNOPT: SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization, SIAM Journal OF Optimization, vol 12, part 4, pages 979-1006, 2002.
24. Bruce A. Murtagh, Michael A.Saunders, Philip E. Gill, Ramesh Raman, Erwin Kalvelagen, MINOS: A Solver for Large-Scale Nonlinear Optimization Problems. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.9437&rep=rep1&type=pdf, as of June 25, 2010, 21:50 GMT.

Appendix

Appendix (A) Friction coefficient table

The following table shows the friction coefficients we used for this thesis.

	On dry asphalt	On wet asphalt
Static friction coefficient for a car	1.20	0.80
Kinetic friction coefficient for a car	0.85	0.60

Appendix (B) F1 car features

The following table shows the speed and acceleration limits for typical F1 cars.[2]

	Usual	Extreme (Maximum)
Positive Acceleration	1.45 g (14.2 m/s²)	
Negative Acceleration	4 g (39 m/s²)	5-6 g
Maximum speed	200 km/h (124 mph)	300 km/h (186 mph)
Lateral acceleration	3.0g	5-6 g