Test report

Table of Contents

[Reference generation 2](#_Toc60815362)

[Reference generation under Cartesian 3](#_Toc60815363)

[Reference generation under Frenet 3](#_Toc60815364)

[Testing results Comparison on the two methods 3](#_Toc60815365)

[Cost function 6](#_Toc60815366)

[Coordinate 8](#_Toc60815367)

[Coordinate projection 8](#_Toc60815368)

[Sampling based planning 9](#_Toc60815369)

[Difference sampling rules 9](#_Toc60815370)

[Scenario divisions 10](#_Toc60815371)

[Algorithm 13](#_Toc60815372)

[x(t)-y(t) method 14](#_Toc60815373)

[SL method 24](#_Toc60815374)

[Path smoothing 26](#_Toc60815375)

[Path smoothing method 27](#_Toc60815376)

[Kappa calculation 28](#_Toc60815377)

[referemce 29](#_Toc60815378)

# Reference generation

in this section, we discuss two methods in reference line generation. The reference is defined as the desired trajectory ego vehicle is supposed to follow. The section is organized as follows:

Reference generation can be based on lane and map. For method based on lanes we provide two methods for reference line generation. One is reference-line-generator, and the other is reference-line-generator-frenet. The difference is obvious: the first one is generated under Cartesian coordinate, while the other is generated under frenet coordinate. In terms of realization complex, the first method is much easier, as it can directly get respective y value given x value.

## Reference generation under Cartesian

As long as the two lanes parameters and the total length LEN are given, we can calculate it as follows:

## Reference generation under Frenet

While for the second method, it can be a little bit more complex. We do integration along lane, and we calculate mean value of two points which have the same longitudinal integration. The details can be seen in the code.

Next, we do extreme test on reference line generator of both: only four cases should be tested:

1, lane turning left longitudinally, and planned trajectory turning left;

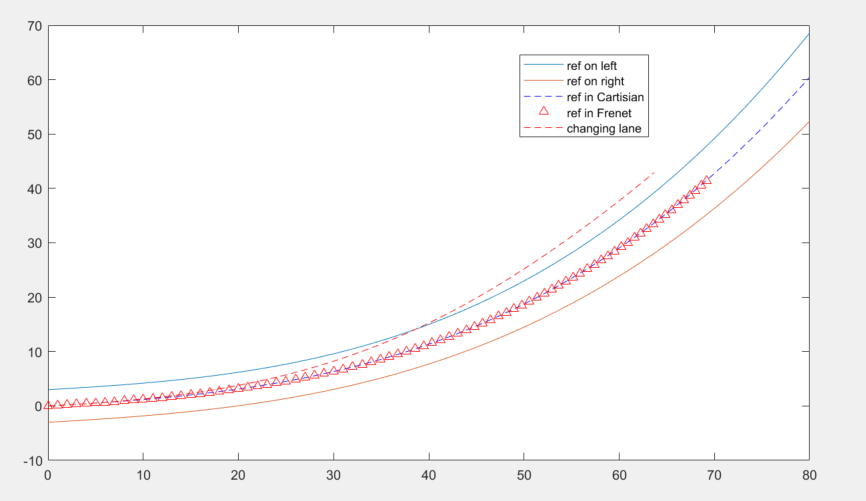
2, lane turning left longitudinally, and planned trajectory turning right;

3, lane turning right longitudinally, and planned trajectory turning left;

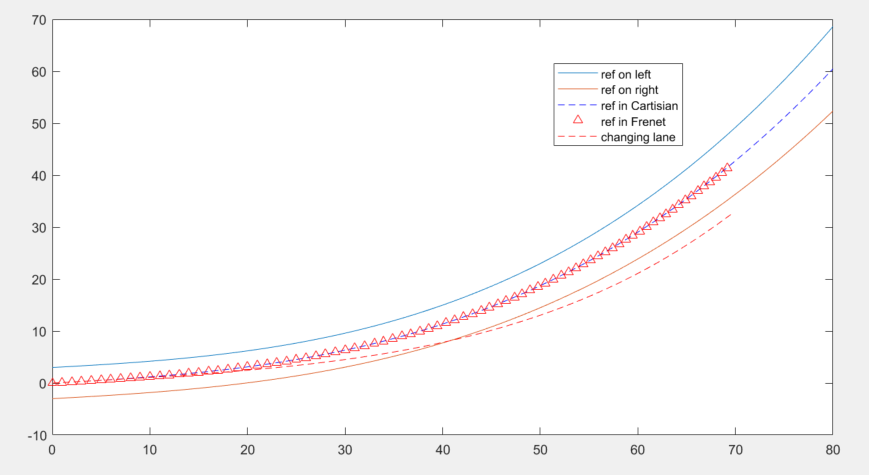
4, lane turning light longitudinally, and planned trajectory turning right;

## Testing results Comparison on the two methods

The test results are as follows:



* 1. Fig.x: Vehicle turning left, with lane changing left



1. Fig.x: Vehicle turning right, with lane changing left

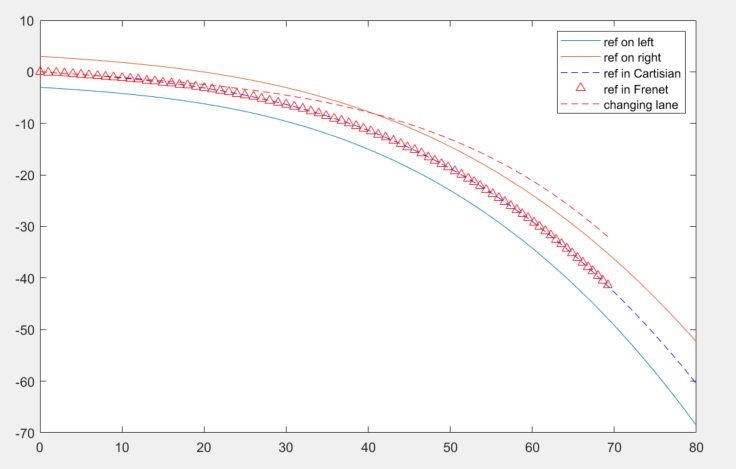


Fig.x: Vehicle turning right, with lane changing left

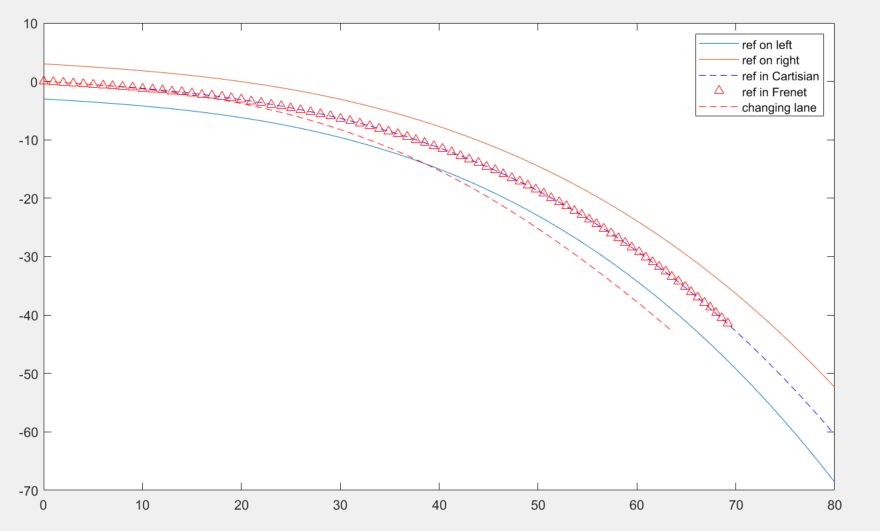


Fig.x: Vehicle turning right, with lane changing right

We have two difference methods in generating a sequence of center line waypoints for future use. One is, based on current polynomials of lanes on the left and right, discretely sample on Cartesian coordinate along x axis to get the y axis value, and calculate its mean value. And the other method is discretely sample under Frenet coordinate, which means it integrate along s direction. At each sample point, get the respective x value on each lane polynomial. We provide both methods realization and make comparison numerically.

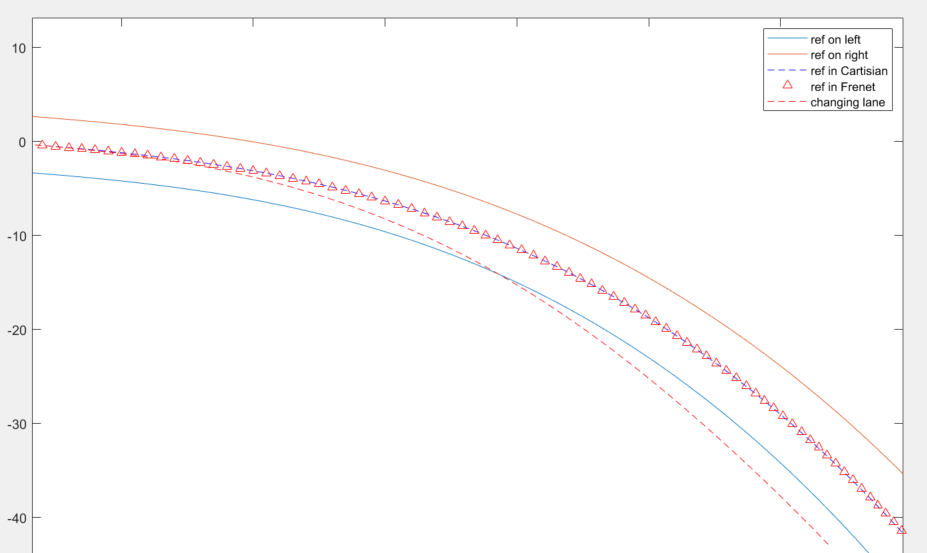


Fig.x: Using Frenet coordinate to generate a sequence of center line waypoints.

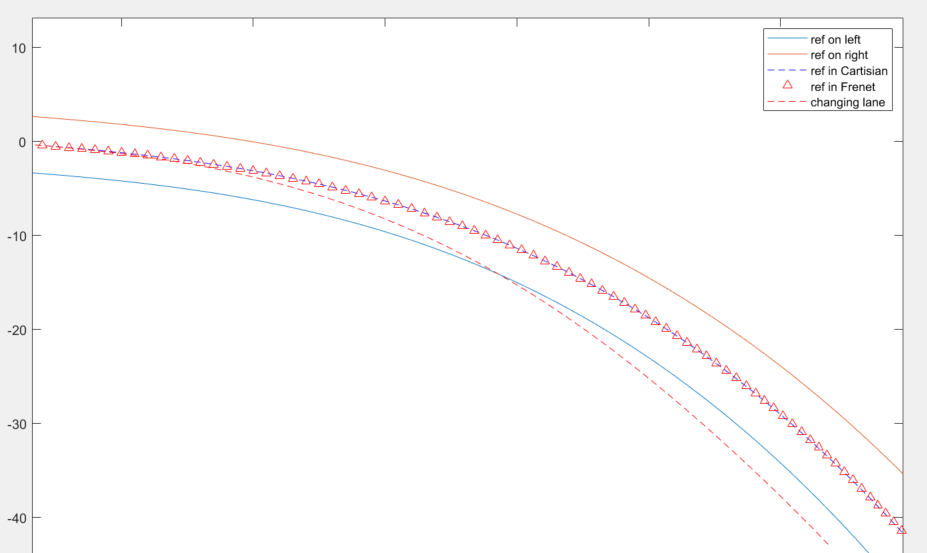


Fig.x: Using Cartesian coordinate to generate a sequence of center line waypoints.

As you can see in the figure, the upper triangle line is using Frenet method, while the blue dashed line is using Cartesian method. They almost overlapped. As for the calculation time, we also make a comparison as follows:

The prerequisite is that both methods generate a sequence of trajectory of 80m, and the sampling resolution is about 0.2m, which means about 400 points will be generated. The calculation time is (with CPU performance i7-9750H @2.60GHz and RAM 32G):

Using Cartesian method:

*reference\_1 generation time is: 1.680400 ms*

Using Frenet method:

*reference\_2 generation time is: 4.659700 ms*

# Cost function

As has been mentioned, we choose sampling-based method to realize our motion planning functionals. The role of cost functions and how it is used will be discussed in the algorithm sections, but we still want to first discuss parameters we choose that consist of our cost functions.

We define cost functions to evaluate the safety (vehicle kinematics), human comfort, driving efficiency, energy consumption, trajectory consistency for each candidate trajectory [11]. We extract *n* points from each trajectory and represent cost function in discrete forms, which are shown in Table 1.

Table 1: Cost Functions

|  |  |  |
| --- | --- | --- |
| Cost | Formula | Physical Interpretation |
|  |  | is path length of each section |
|  |  | is curvature |
|  |  | is first-order derivative of curvature |
|  |  | is second-order derivative of curvature |
|  |  | is third-order derivative of curvature |
|  |  | is lateral offset with the closest reference line |
|  |  | is lateral acceleration |
|  |  | is longitudinal acceleration |
|  |  | is rate of change of |
|  |  | is rate of change of |
|  |  | is distance between current trajectory and the reference line, is distance between previous trajectory and the reference line |
|  |  | is time duration of a trajectory |

Next, we are going to discuss how these cost functions affect the curves.

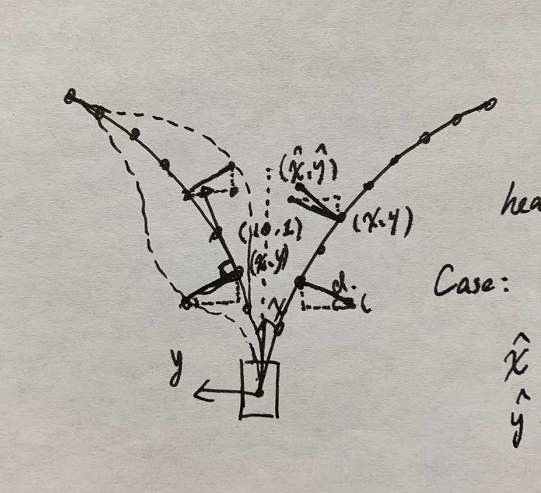
………….

# Coordinate

Here, we want to mention how to make transformation from global Frenet coordinate to vehicle coordinate. In this section, we want to use Bezier curve as trajectory generation method as basics. The transformation consists of two parts: 1) generate under Cartesian coordinate the planned trajectory. If it is a Bezier curve, the only two parameters we need is its terminal longitudinal and lateral distance w.r.t reference line ,. Then, with reference line generated by current lane marks under vehicle coordinate, we move the reference line points perpendicularly to corresponding longitudinal integration distance s. Such moving distance is searched from planned trajectory at corresponding s. under Frenet coordinate, …………………………………

## Coordinate projection

No matter what sampling method we use, we always have the same output from the motion planner core algorithm. It outputs reference line data and planned trajectory data that is presented by lateral offset with respect to longitudinal distance s along reference line. If it has velocity profile, it is also given as the same way. The reference data is given under vehicle coordinate, while the lateral offset data is given under Frenet coordinate, therefore, we do a projection to merge them together.



Fig,.x: ….

The projection rule is as follow:

)

# Sampling based planning

In this section, we want to discuss how we use sampling-based method in our design. This section is organized as follows: 1) we discuss difference rules in do sampling-based planning. 2) we divide on-road planner scenario by scenarios and introduce different sampling method in dealing with these scenarios. 3) we specifically discuss how to use sampling method. 4) we introduce a simplified trajectory generation method to help realize Lane Changing Assistant (LCA). 5) we discuss how to make sampling more efficient. 6) we discuss how to choose cost function terms.

## Difference sampling rules

In sampling-based method, we should always obey the rule that we combine two dimensions at a time. For example, we can sample x(t) and y(t) and combine them together to get spatial-temporal profile. By doing derivative over time the velocity profile can also be generated. Another method is to sample y(x), and x(t), which we named more formally as “SL and ST graph”. The y(x) is the SL graph, which indicates the relationship between lateral position and longitudinal position. The x(t) is the ST graph, which indicates the relationship between longitudinal position s and time t. In terms of sampling dimensions, SL samples on lateral offset L and longitudinal S, and ST samples on time T and longitudinal S. The specific methods will be discussed later.

Another sampling method, as mention above, is to sample lateral offset x and time t, and longitudinal s and time t, say x(t) and y(t). let us compare between these two sampling methods. As for the first method, SL and ST graph method, can be applied more specific kinodynamic constraints (which mean considering vehicle kinematics and dynamics). The reason is that we can calculate on SL graph the curve curvature, jerk (spatial continuity), etc. while in its counterpart, x(t)-y(t) sampling method, we only have separately spatial temporal profile laterally and longitudinally, which means it is more decoupled in the two directions, therefore we are less likely to apply kinodynamic constraints on the curve. The only thing we can do is to check it feasibility at the very end of the trajectory generation process. More specifically, we can only do velocity, acceleration, and curvature check when the two graphs are combined to together. Since feasible velocity and acceleration in each direction does not guarantee that they are feasible after combining. And curve curvature cannot be calculated until combining. This method, however, also has its merits. Since it is sampled in lateral and longitudinal over t, it is more sensitive to deal with dynamic obstacles, for example, merging situation. When we have time information in both dimensions, sampling can be more organized. We can have unified T sampling range for both x(t) and y(t), since the two directions are decoupled. Even though SL-ST method samples time in ST graph, but since we have longitudinal sampling in SL graph, for different sample S, we have to first calculate a feasible T duration, so that it can be realized longitudinally. Therefore, In ST graph sampling of T, it is more uncertain. Uncertainty always decrease robustness. The advantages of x(t)-y(t) method is obvious, while its weakness, as mentioned, is also obvious: decoupling lateral and longitudinal sampling means neglecting vehicle kinodynamics. Postponing status checking which ensures trajectory feasibility always leads to low efficiency. As you need to check on each candidate curve for its feasibility, and calculation efforts are wasted until you find the sub-sub-…sub-optimal trajectory that satisfy the status checking.

## Scenario divisions

we decide to divide our target scenarios into several classes so that we use difference sampling methods to enlarge their advantages at most.

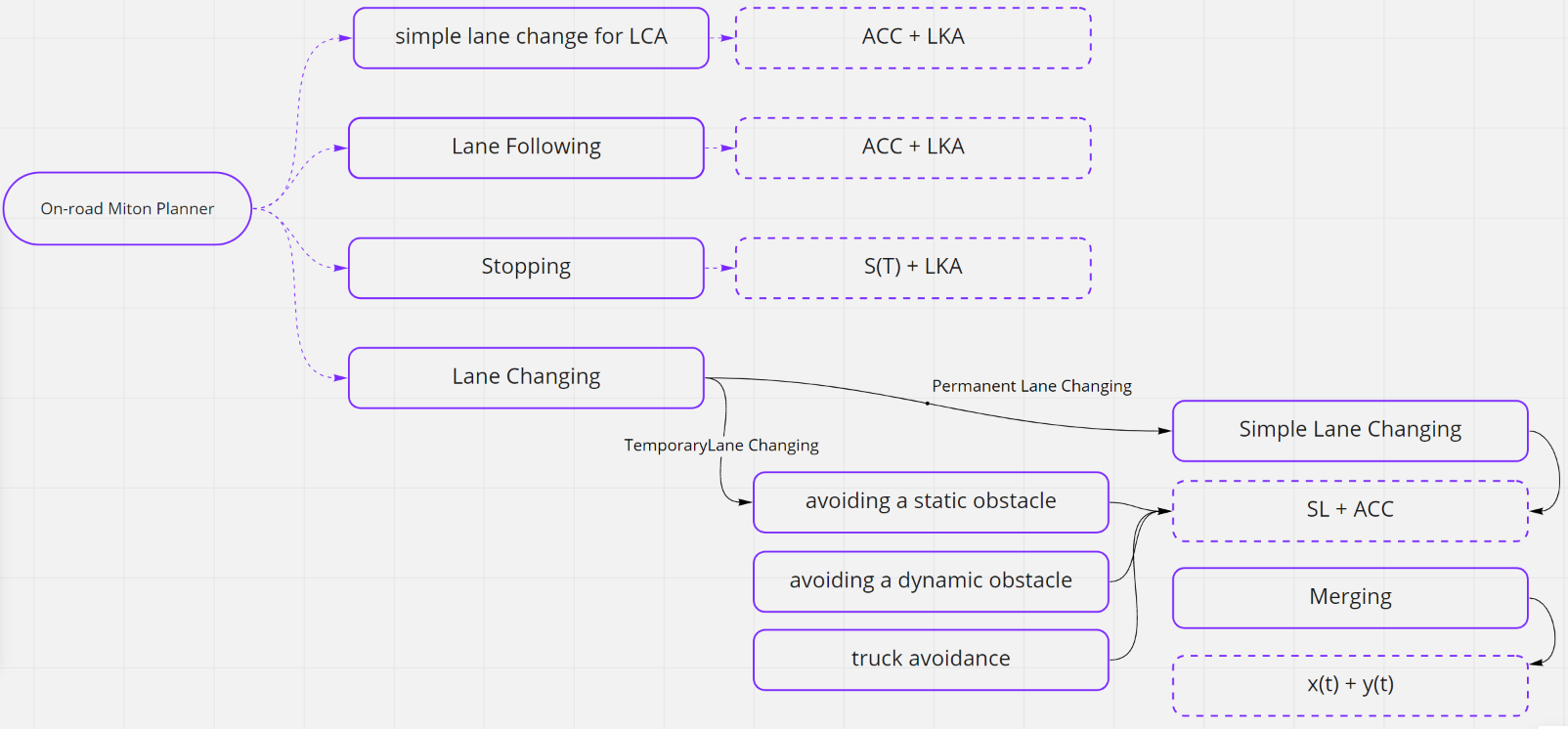


Fig. x: We divide the on-road driving into several scenarios, and in each scenario, the motion planner can be slightly different.

1. Lane following

This scenario deals with only lane following without lane changing, it is further separated into velocity following and distance following. Since this is the simplest scenario, we want to simplify planning process as much as possible, even in longitudinally planning, we give up sampling method. Instead, we use trajectory tracking method longitudinally, with reference line (usually rough lane centerline is provided by mapping module, and post-processed by smoother module). Such decide at most reuse the merits of LKA and ACC functionals.

1. Stopping

As for stopping scenarios, for example, stopping in front of the red traffic lights, or in front of some static obstacles appearing in front of the lane, etc. we create a virtual stopping line where the vehicle must stop to deal with this scenario. Laterally, we still use the same method as lane following, assuming that we will not allow sudden steering when the vehicle is in process of stopping. Longitudinally, we apply s(t) method, where we sample on longitudinal distance s and time t, to get a profile. By doing derivative over time, we get relationship between longitudinal distance s and time t.

1. Lane changing (permanent lane changing & temporary lane changing)

Even for lane changing scenarios, we still divide it into several more specific situations, say, lane changing whose target lane is still its original lane (temporarily changing its lane, for example avoiding a static or dynamic obstacle in front of ego vehicle’s current lane, truck avoiding, etc.) and lane changing whose target lane is some adjacent lane (permanently changing its lane, for example manual lane changing, automatic lane changing, merging, etc.). The separation criteria is that we use different trajectory generation methods: for example, simple lane change uses Bezier curve as basic trajectory, while merging uses quintic polynomial trajectory generation method. We also have different cost function terms and cost function parameters in the two methods.

For the first situation, we apply SL-ST method. Since we do not have too much velocity interaction with other vehicles. The most important issue in dealing with such scenarios is lateral planning, we still can use original ACC functional to support its longitudinal velocity planning, as we do not care too much about at what exact timestamp, we reach the lateral position. We only care about that the vehicle can at some time during go through the lateral positions as we have previously planning. In dealing with static and dynamic obstacles, and temporarily shifting the reference line a little bit to avoid adjacent trucks, such method works well. In terms of computational efforts, this method only sample in longitudinal s and time t, which means it decreases the sampling dimension from 3 to 2, greatly reduce the computational efforts. ACC and LKA always uses less computational effort that sampling based method; thus, our principle is that we use ACC and LKA so long as the scenario is simple enough to support such functional combination.

For the second situation, where vehicle permanently change its lane, we also divide it into two different situations. Let us assume that the selected vehicles on the target lane are vehicle A in the rear and vehicle B in the front. The selected vehicles on ego lane are vehicle C in the front and vehicle D in the rear as shown in Fig.x.

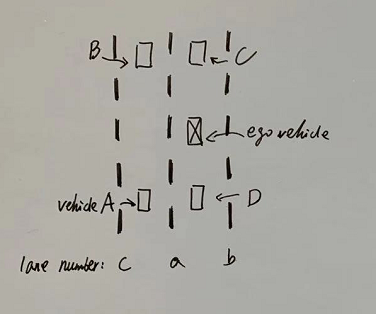


Fig.x: vehicle labels

We give case number to different scenarios; four cases are divided into two groups where x(t)-y(t) method and SL method are used.

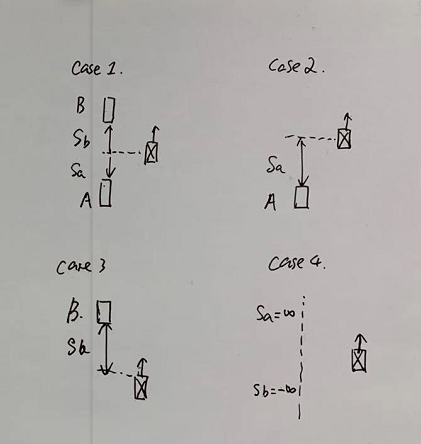


Fig.x: …

In case 1 and case 3 scenarios, x(t)-y(t) method is used, while in the other two cases, SL method is used. The reason is that in the first group, the vehicle velocity is affected by either the two vehicles or the front vehicle. While in the other group, the vehicle velocity is free for control. But one special point in case 2 is that the planned velocity of ACC cannot drop below to velocity of vehicle A.

For simple lane changing, we plan laterally using SL graph, and ACC plans longitudinal velocity. While for merging situation, which is the most complex so far, we use x(t)-y(t) method. You may feel that we can still use SL-ST method in such situation, but the answer is NO. Imaging that we need to cut into the middle position of two adjacent vehicles. We have to calculate the exact which is a function of t, and if we first planning the SL graph where we get d(s) (d is lateral offset), actually, we have fixed the time t. If we sample on t, the varies, but in ST graph, we want to sample on both S and T. As mentioned, we have fixed t in SL sampling, we do not have any freedom in sampling on t in ST graph. To sum up, in dealing with merging situations, if we try to use ST-SL method, one dimension is always fixed due to their coupling relationship.

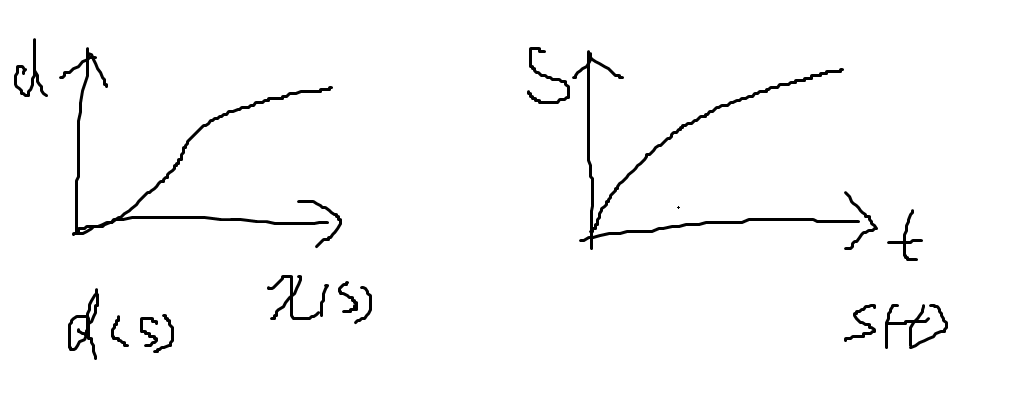


Fig. x: In merging scenario, one dimension is always fixed due to the coupled relationship between the three dimensions.

Now it is obvious that x(t)-y(t) method is the only choice we have in dealing with merging. We first sample on d (lateral offset), and then sample on T, once T is determined, we have , the longitudinal s sampling is around the . In this method, the three parameters, x, y, t is three free dimensions, and no dimension is fixed due to their coupled relationship.

## Algorithm

a). Automatic Lane Change

b). Manual Lane Change

c). Avoiding Static Obstacles

d). Avoiding Dynamic Obstacles

e). Avoiding Truck

f). Stopping

g). Lane Following

### x(t)-y(t) method

The most complex on-road scenario in this report is merging, as we consider both longitudinal and lateral planning at the same time. In other cases, we can simplify scenarios to decrease sampling dimensions from 3 to 2. In this case, however, it seems that we cannot avoid sample on all three dimensions at the same time. As for choice of sampling method, as mentioned in previous section, we must use x(t)-y(t) method. Based on [Werling,2010], we apply very similar merge scenario sampling.

In lateral movement design, we plan x(t) under instantaneous vehicle coordinate. Human perception obviously weights lateral and longitudinal changes of acceleration differently. We introduce jerk in both longitudinal and lateral directions in Frenet coordinates, which are denoted by and . From [xxx] we know that quintic polynomials are the jerk-optimal connection between a start state = [, , ], and an end state = [, , ] within the time interval T in a one dimensional problem. More precisely, they minimize the cost functional given by the time integral of the square of jerk

Since we seek to minimize the squared jerk of the resulting trajectory, we choose the start state of our optimization = [,, ] according to the previously calculated trajectory ***s***, so that no discontinuities occur. For the optimization itself, we let  *= = 0*, as we always want to constrain the terminal states, which is lateral velocity and lateral acceleration, to zero. In addition, we choose and end state lateral offset d as another two cost function penalty items. The cost function is denoted by:

,

where ,, and is respective penalty weights on different parameters. is the mean value along the planned trajectory, T is planning time T, and is mean value of squared lateral offset of the planned trajectory along the reference line.

Instead of calculating the best trajectory explicitly and modifying the coefficients to get a calid alternative, we generate in the first step a whole trajectory set: by combining different end conditions:

*=* []*,*

where we sample on end state of lateral offset ***d***, and planning time ***T***. as we have mentioned in previous section that x(t)-y(t) method does not support curvature calculation, therefore, we only choose three penalty items in cost function. Status checking such as velocity checking, acceleration checking, curvature checking, and collision checking have to be postponed until we generate respective longitudinal trajectory x(t). after we combine the x(t) and y(t) trajectory and get the trajectory with lowest cost function, we check the trajectory against velocity, acceleration, curvature, and collision. If we are luck, it is valid, and we are done. If it is not, we would have to pick up a “sub-optimal” trajectory with the second lowest cost function, and do status checking again until we find a collision free alternative.

For any combination of and . Since the quintic polynomial has six constraints: = [,, , ] and , we can get a close form solution for the polynomial parameters. At the starting point , we have:

At the end point , we have:

For convenience, let’s always assume that . The six parameters can be solved as follow:

,,and can be achieved by solving the following matrix function:

where

.

In longitudinal movement design, we plan y(t) under instantaneous vehicle coordinate.

Since distance keeping, merging, and stopping at certain positions require trajectories, which describe the transfer from the current state to a longitudinal, possibly moving, target position , we generate a longitudinal trajectory set, analogously to lateral trajectories, starting at = [,, ,] and vary the end constraints by different , and according to

*=* [].

The cost function is denoted by

where ,,are penalty weights, respectively, on jerk, planning time, and longitudinal distance offset between sampling distance and .

Let us define that merge scenario is to merge into two consecutive vehicles *a* and vehicle *b* on adjacent lane of ego vehicle. We reasonably assume:

Time integration leads us to

，

where .

In merging scenarios, we define

,

while in real testing cases, we find that it can also be defined as:

.

We may only use the two regular equation for the next deduction, but since we have divided the cases of permanent lane changing into several scenarios, it is necessary to give another two scenarios calculation. For scenarios that we have only vehicle in front direction, ego vehicle needs to merge at the back of the front vehicle and keep some distance in the end, we have:

For another scenario where we only have back vehicle, and ego vehicle needs to merge in front of the back vehicle, we have:

In real situations, we may give up the last one, as we do not care about the vehicle velocity behind when merging. Therefore, we still give only spatial trajectory planning in such situations. Velocity is control by ACC.

Let us go back to the deduction of the first two main forms. For the first form of , time derivatives of is as follows:

If we make a simpler assumption that

Time derivatives of is as follows:

For the second form of , time derivatives of is as follows:

If we make a simpler assumption that

Time derivatives of is as follows:

Now the end point boundary condition is determined, and the longitudinal quintic polynomial can be solve using the same method as lateral quintic polynomial solver. The starting point boundary condition is specified as:

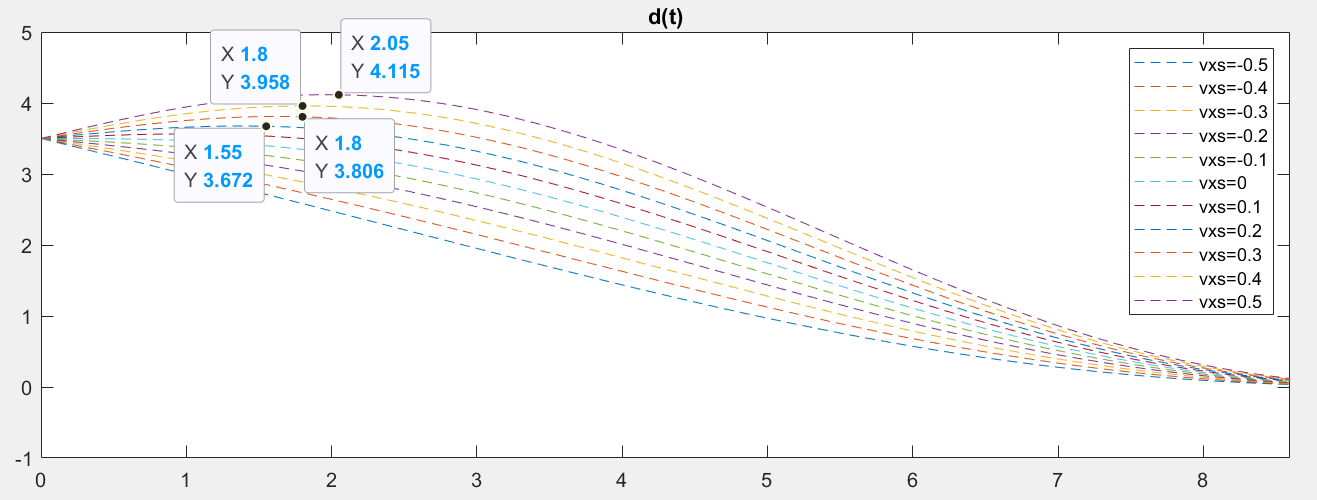
= [,, ,] = [0, , , 0]

We may have a deeper look at the quintic polynomial planning.

In lateral direction, we use ***test\_on\_lat\_poly.m*** file to test the polynomial reasonability. We give different initial value and boundary condition to see whether it is feasible trajectory that satisfy vehicle kinodynamics.

The most obvious problem we may have is bad initial lateral velocity and acceleration can lead to bad curve. In the following section, we give some examples of different cases with different initial and end states.

In the first case, we only give freedom to initial lateral velocity (in codes it is denoted by ***vxs***). Let us make it vary between -0.5 to 0.5 with step 0.1. due to its initial lateral velocity that is against the lane changing direction, the trajectory will initially move to the opposite direction of the lane changing direction, and the effect is even stronger as the initial lateral velocity increase. Therefore, a large lateral velocity may bring about more difficult to lateral control.



From the code, we see that velocity ranging from 0.1 to 0.5 brings maximum lateral offset as [3.5648,3.6759, 3.8100, 3.9584, 4.1166].

What if we constrain the lateral velocity to a slightly low value 0.3m/s, and see how T affect the polynomial? In this example, we range T from 5 to 10:

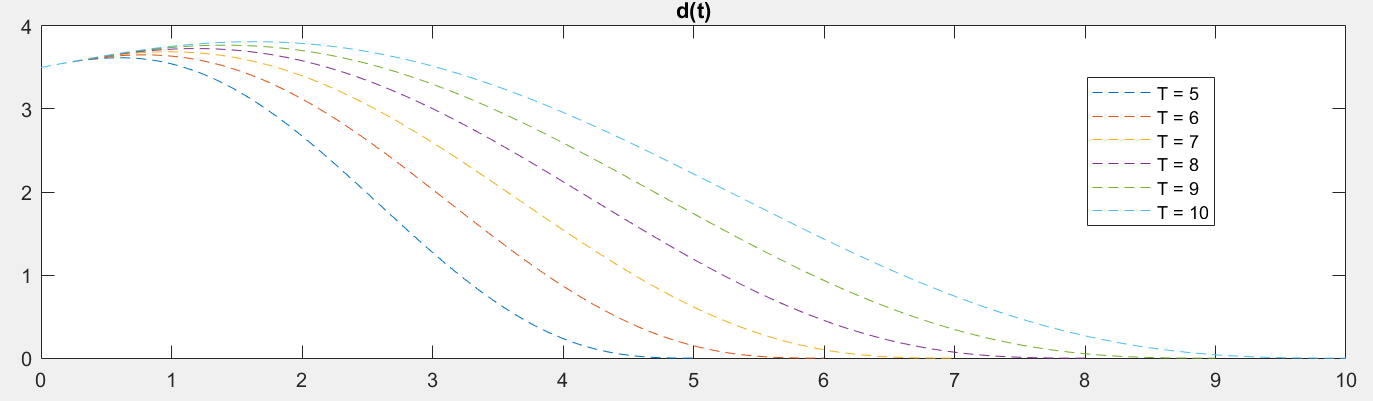


Fig.x: we can see from the figure that longer time T may increase the maximum lateral offset given the same initial lateral velocity.

Another effect is lateral velocity minimum. If the initial lateral velocity is against the lane changing direction, we may have problem of inverse lateral velocity overshot.

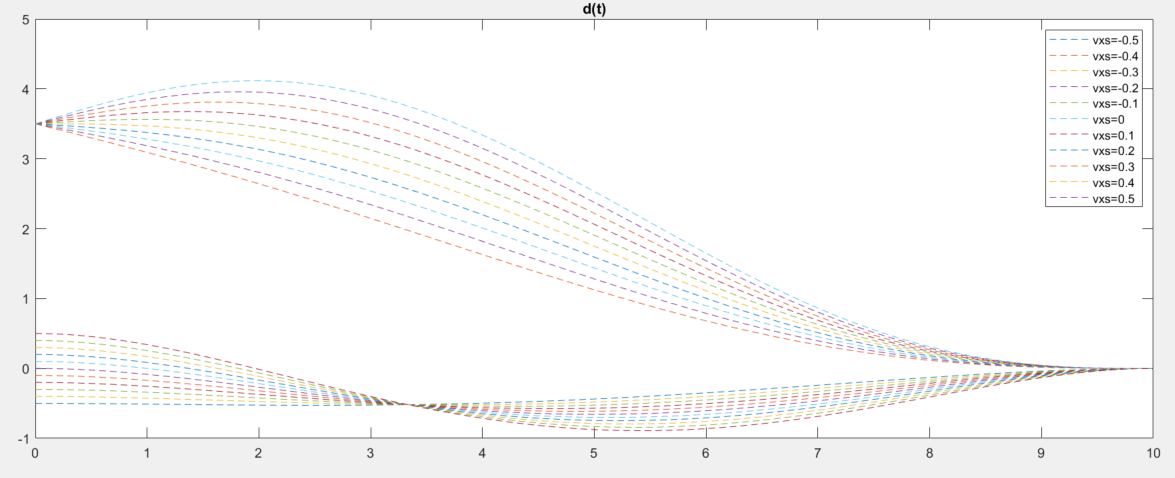


Fig. x: high initial lateral velocity leads to inverse velocity overshot. In this figure, we can see that with initial velocity 0.5m/s, the minimum lateral velocity reaches almost -1m/s.

Therefore, we decide to simplify initial condition, namely, constraining the lateral velocity in some reasonable range, so that lateral position overshot, and inverse velocity overshot can be limited to a reasonable range. The initial lateral velocity is limited below the absolute value of 0.3m/s.

In the next case, let us see how the terminal lateral offset affect the polynomial generation. We still constrain the initial lateral offset to a slight lower number 0.3m/s and sampling time T = 8s.

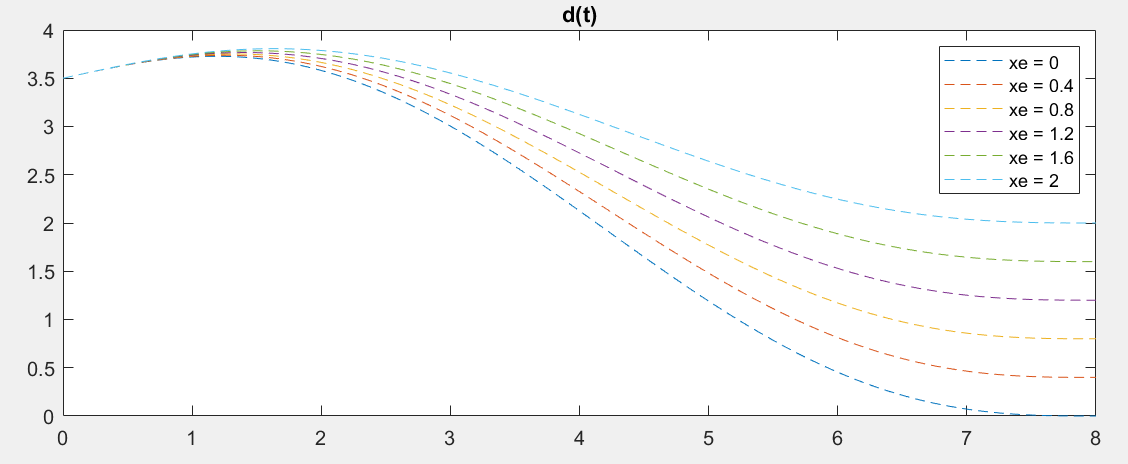
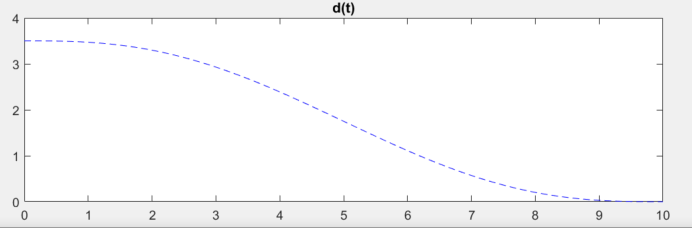
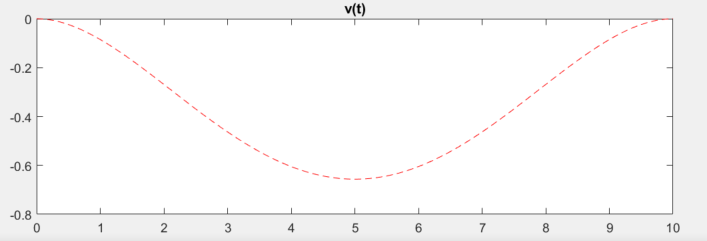


Fig.x: we can see that sampling a large terminal lateral offset can lead to large maximum lateral offset.

We also make some other preposition that the initial lateral acceleration is always 0, and the terminal lateral velocity and acceleration are always 0.

As for how to choose a reasonable time T, it is still a empirical issue, we give simple example with initial lateral offset 3.5m, time T = 10s, and other conditions zero. The simulation results are as follows:





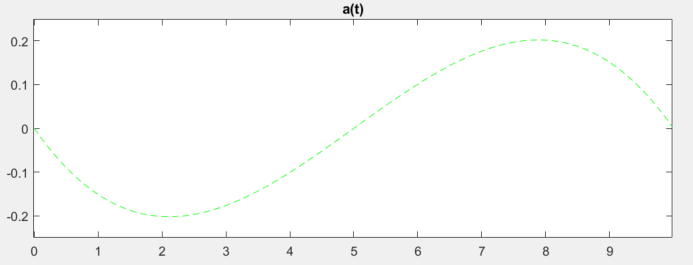
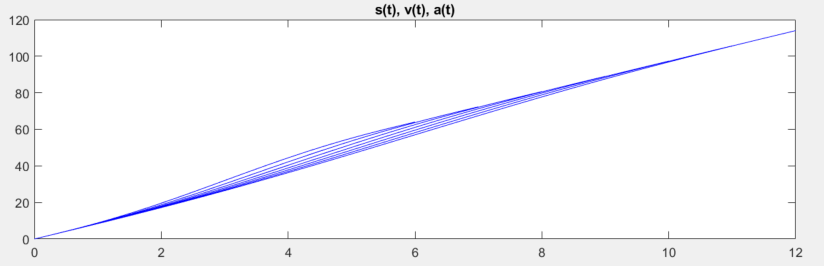
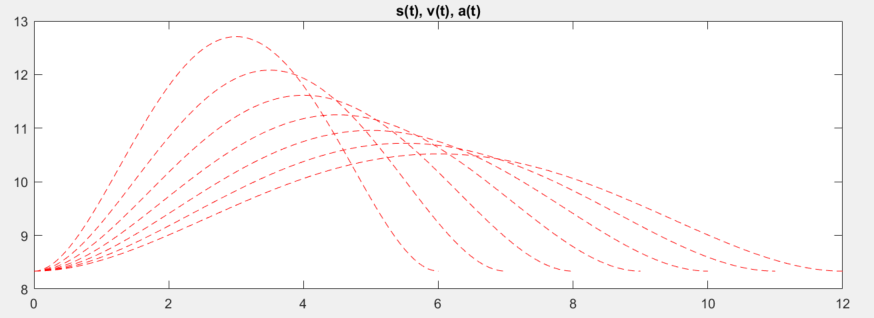


Fig. x: it is obvious that choose a reasonable sampling time T and given reasonable initial conditions can lead to reasonable polynomials and velocity, acceleration profiles.

In longitudinal direction, we do the same boundary condition tests on time T.





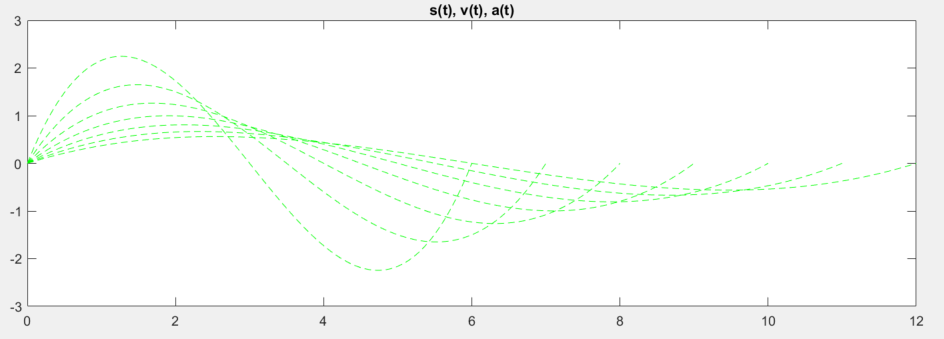


Fig. x: in this figure, we give initial boundary conditions as adjacent vehicles move at the same speed of 30kph, and ego car is at the parallel position as the closest vehicle. The distance between the two adjacent vehicles is 21m. we sample on time T between 6s and 12s.

At least in this scenario, the sampling time range is reasonable. We check the longitudinal velocity and acceleration to see weather it is reasonable, we declare that the maximum velocity cannot reach over maximum road velocity limit, and the acceleration cannot reach over maximum and minimum acceleration limits.

The single function is realized in \laneChangeFunctional\polySamplingMerge\ location, and file name is ***test\_on\_module.m***. In this file, we provide basic functional realization for any related scenarios. The details of how the function is realized can be seen in the comments in the codes. Here we provide the simulation results. In this test case, we use

Initial condition: = [,, ,] = [*0, 10, 0, 0*]; = [,, , ] = [*-2.5, 0, 0, 0*]. The details of sampling can also be seen in the code comments. The results are shown below:

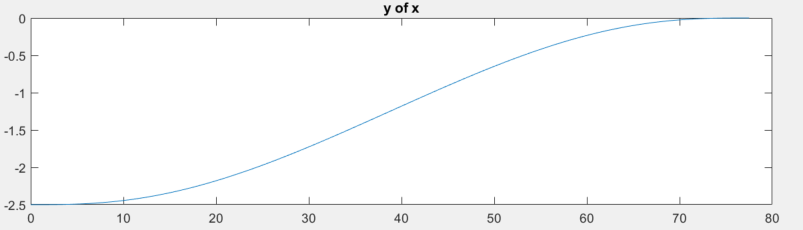


Fig. x: y of x plot

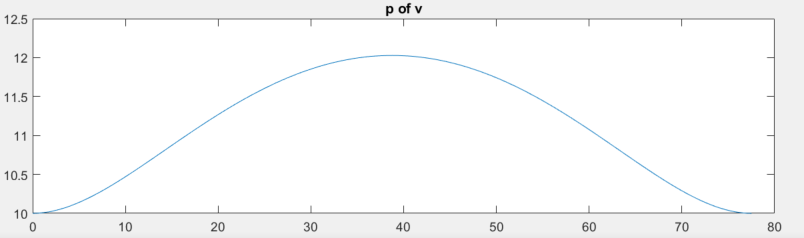


Fig. x: p of v plot

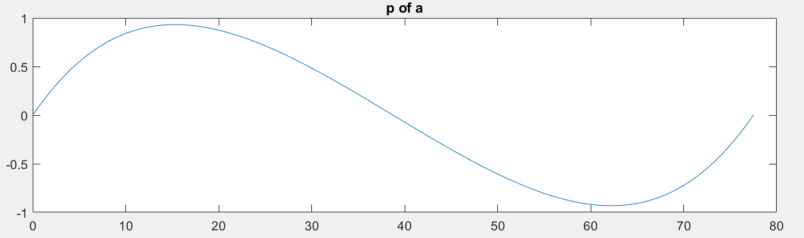


Fig. x: p of a plot

It can be seen in the above three figures that under frenet coordinate, the planned trajectory in one loop is shown move from initial position [0, -2.5] to [80,0]. As for the velocity and acceleration profile, the minimum and maximum limits are satisfied.

The above test does not consider sampling and optimization, now we do sampling and find the satisfactory trajectory with lowest cost function. Same thing for transformation part, the only different is how to find the optimal trajectory. We do sampling on T, S, and D, and get all possible combination of the three parameters, sort the combination by cost from low to high. Then do status check to see whether the trajectory with the lowest cost function can satisfy status check, if it is feasible, it is done. If it does not satisfy the criteria, use the trajectory with second lowest cost function(sub-optimal) to do status check. We do such status checking until we find the feasible trajectory. If we cannot, finally, find the feasible optimal trajectory, it is defaulted that we use the trajectory with the lowest cost function. This part is still an open challenge, the appropriate way is to design a backup planner for such corner case, or in behavior planner, we in advance find a feasible trajectory for backup. But still, there are a lot of work to do to deal with this case. We do another case test with caseNum = 1, which means the vehicle mering into two vehicles, the initial conditions are:

= [,, ,] = [*0, 10, 0, 0*]; = [,, , ] = [*-6, 0, 0, 0*],

,.

and the simulation results are as follows:

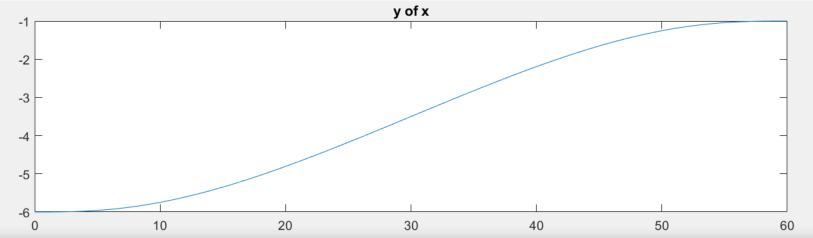


Fig.x: y of x

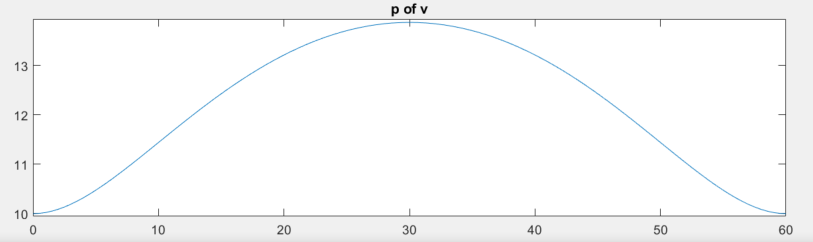


Fig.x: p of v

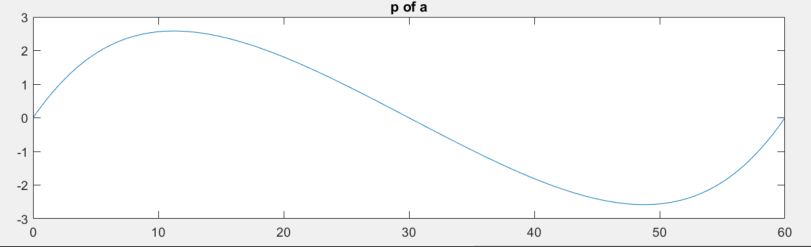


Fig.x: p of a

After transformation, we plot the planned trajectory in vehicle coordinate.

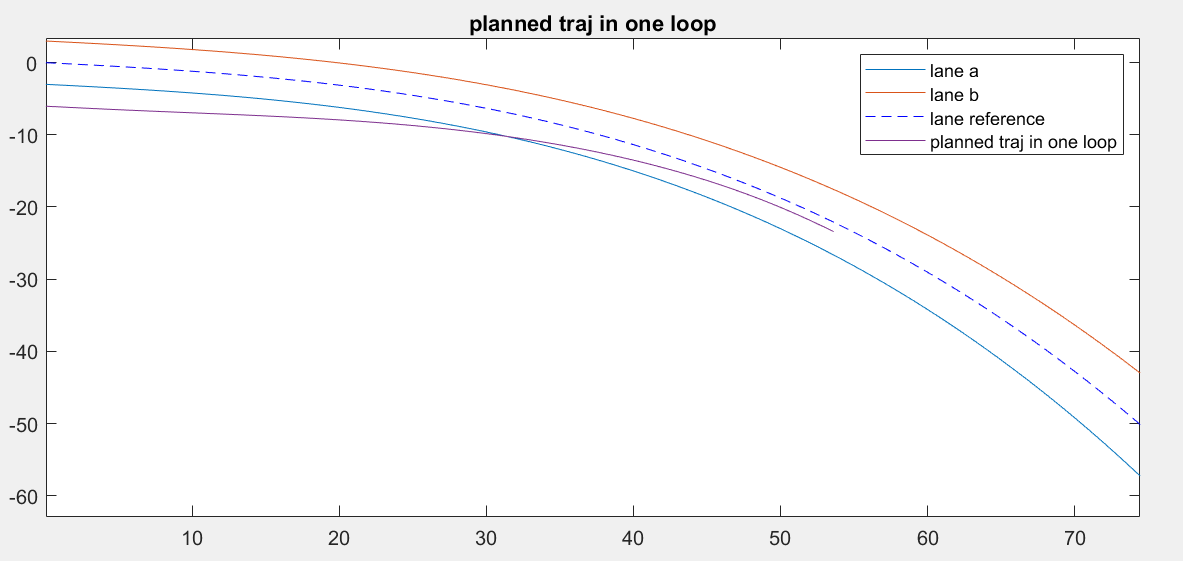


Fig.x: planned trajectory in one loop

### SL method

We create a new scenario test in this file with initial lateral offset -6, and the current lane polynomials are given. The test result is as follows:

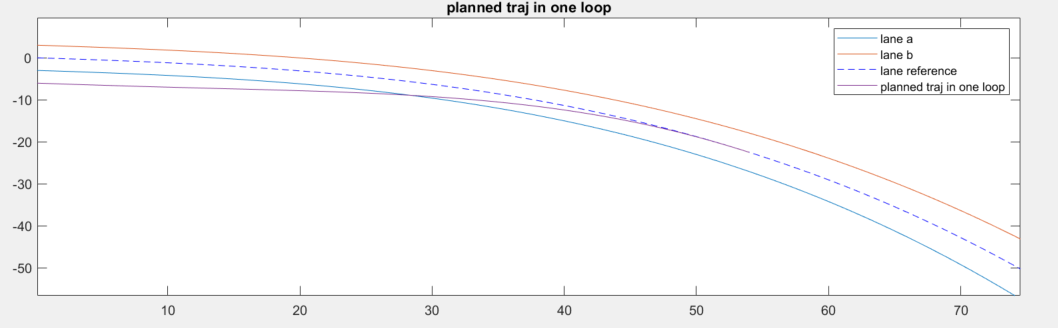


Fig.x: in this figure, the current lane and centerline(reference) are plotted with planned trajectory in one loop in the same figure to see how the planned trajectory looks like.

In this section, we discuss our method of candidate trajectory generation method which is based on lattice sampling (kind of), notice that we do not precompute the lattice heap matrix. In our functionals, this is realized and simulated in frenet\_Bezier\_Sampling.m file, and we also provide a simpler functional where a Bezier curve is individually generated once for Lane Change Assistant (LCA) use which is realized and simulated in frenet\_Bezier\_simpleRule.m.

Following is a test on the robustness of using quintic polynomial to do longitudinal trajectory planning. The test file named as *test\_on\_long\_poly.m* is the simulation code for testing such polynomial in longitudinal direction. By changing the fixed boundary conditions, we can plot the longitudinal profile of x(t), v(t). combining these two figures together, we can get s(v), and when spatial-temporal profile is given, we can search respective v at specific s in the s(v) graph. As we mentioned above, we do not use such method in dealing with merging tasks, but it works well in other scenarios.

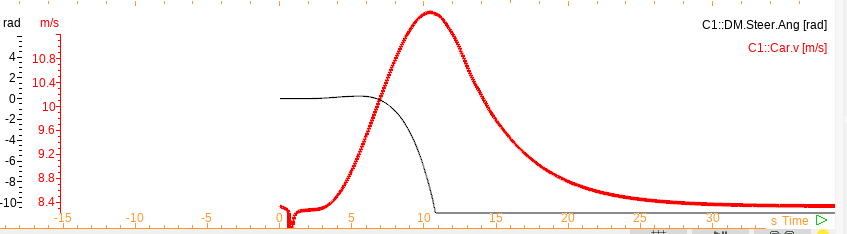
We use the file to do a simple test in test\_on\_laneChanging.slx. The assumption for this test is as follows: we have two adjacent lane vehicles on the right-hand side moving at the same speed as ego vehicle, which is 30 kph. The ego vehicle has the same longitudinal distance as the first neighbor vehicle that is closest to ego vehicle, and the other vehicle is about 50 meters ahead of the first neighbor car. The test scenario is that the ego vehicle is going to merge into the area between the two neighbor vehicles. This test file can generate a speed profile alongside the longitudinal *s*, and it can be used in *test\_on\_laneChanging.slx* file for Carmaker simulation. This test file has not added sampling-based method longitudinal planning functionals and it also does not support such functional simulation. Therefore, it can only test on very specific situation, say, fixed boundary conditions. The merits of this test file are a brief view of how quintic polynomials performs in longitudinal planning and have a look at the changing lane functional performance. Later, when sampling-based functionals are developed, the sampling strategies can also be tested under the similar framework.

The result of this test file is shown in the following figures.





In merging scenarios, the most important thing is velocity control, therefore, a precise velocity tracking controller is critical, it can greatly affect the system performance in merging maneuvers. The following figure is the velocity and steering angle during the merging process. We can see that the ego vehicle velocity increase first and then decrease. Since we only have a simple test with boundary condition fixed, we can not penalize high acceleration during merge, actually, we do not have any control on the duration acceleration. That is why we need to introduce sampling-based method so that we can penalize high acceleration through sampling and cost function minimization.



# Manual Lane Change

## Algorithm

In solving a simplified cost function minimization problem, we may introduce quadratic programming method. The standard QP(quadratic programming) problem is formalized as follows:

The presumption of current cost function minimization is that we constraint the sampling dimension T in the aforementioned sampling problem. Now we use lateral planning as an example to show how to solve it using QP solver.

The standard form of a quintic polynomial is:

The cost function is defined as:

where is fixed sampling time, is jerk of lateral position which is third order derivative of .

Rewrite it in matrix form:

Same thing for jerk calculation:

H matrix is:

The inequality constraints are the initial position, velocity, and acceleration:

For convenience, we always set , and .

Rewrite them in matrix form:

Combine them together we get:

The inequality constraints are that the terminal lateral positions are constrained in a reasonable range defined by lane width:

where and are lower bound and upper bound.

Rewrite it in matrix form, we get:

\

## Path analysis

In this section, we compare how difference parameters affect the final results of the path generation. In test file ***comp\_on\_quintic\_lc***, we make comparison between different parameters combination. We choose velocity varying from 36kph to 108 kph, and lane changing time varying from 3s to 10s, we plot the generated trajectory kappa and lateral acceleration to see whether it is reasonable, and we also want to find a reasonable range of lane changing time that can generate well shaped trajectory. The result is as follows:

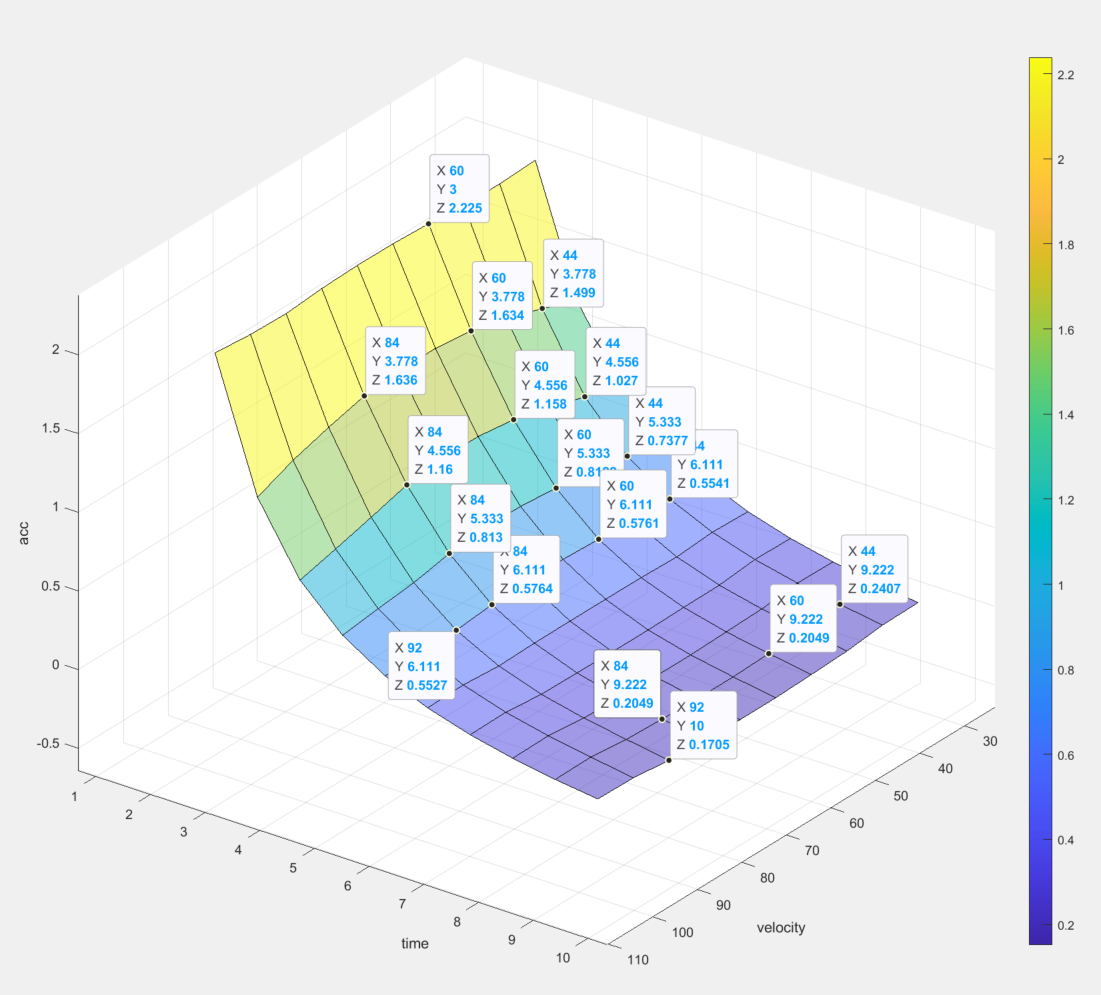
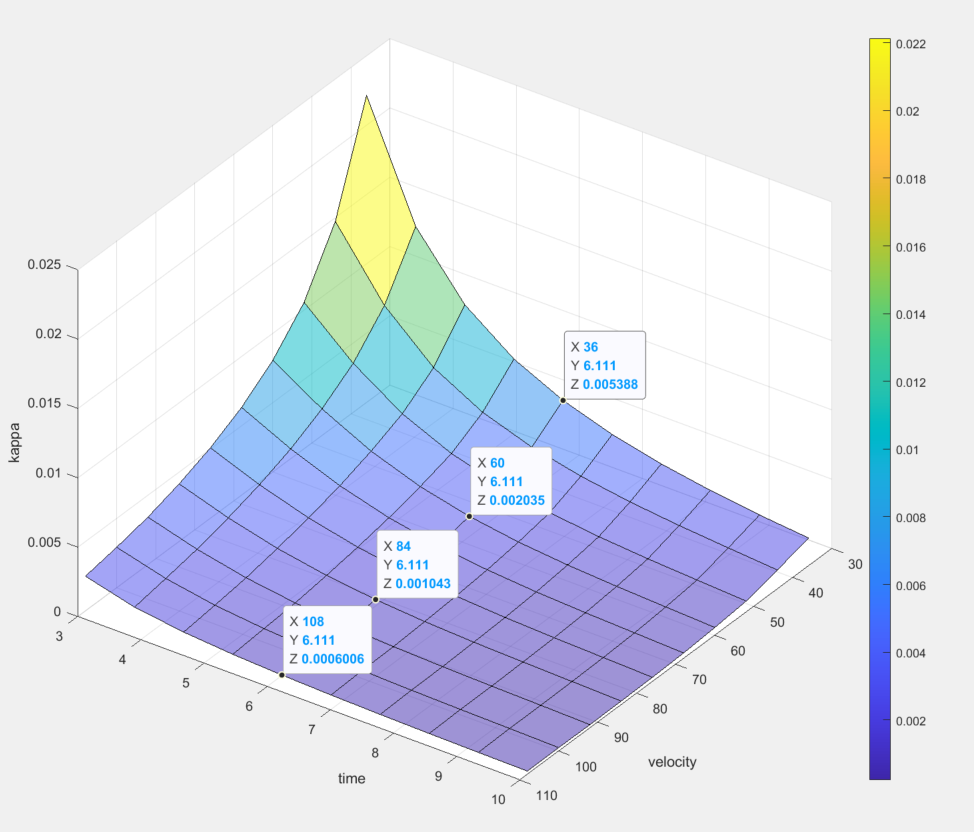


Fig.x: lateral acceleration profile

From the figure, we find that lateral acceleration is greatly affected by lane changing time, while it is only slightly affected by velocity. The reason is that when calculating longitudinal distance, we have considered the time and velocity. Therefore, the time variable has very little effect on maximum lateral acceleration. On the other hand, the most effective variable is lane changing time t. by constraining maximum lateral acceleration, we can find reasonable lane changing time from the figure. Same thing for maximum curvature. For example, if we constrain maximum curvature at 0.005, we can also find the reasonable lane changing time range from the figure.



# Path smoothing

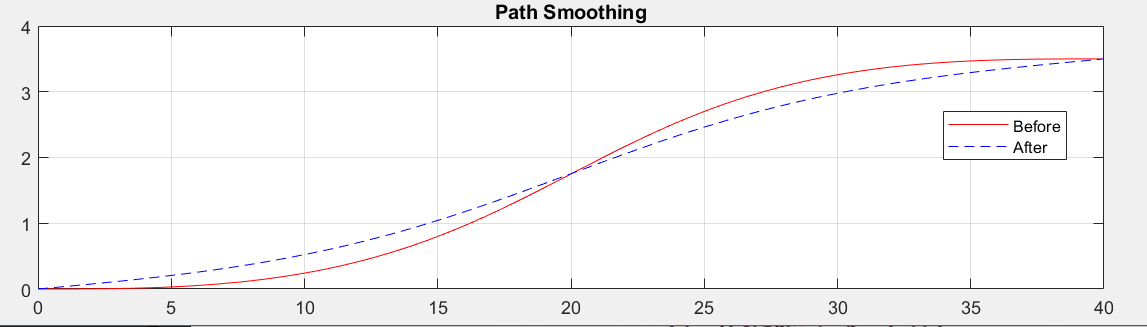
## Path smoothing method

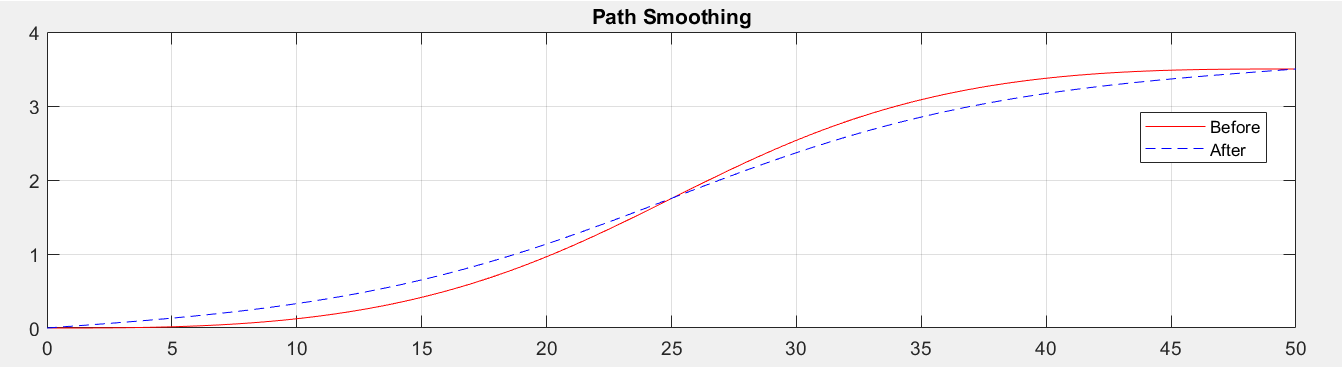
Since the original trajectory generated by Bezier curve cannot guarantee that the curve can satisfy the vehicle kinematic constraints, say, the max curvature constraints. Therefore, we designed a path smoother which aims to minimize 1) the distance between the origin reference line and the optimized trajectory. 2) the smoothness of the optimized trajectory.

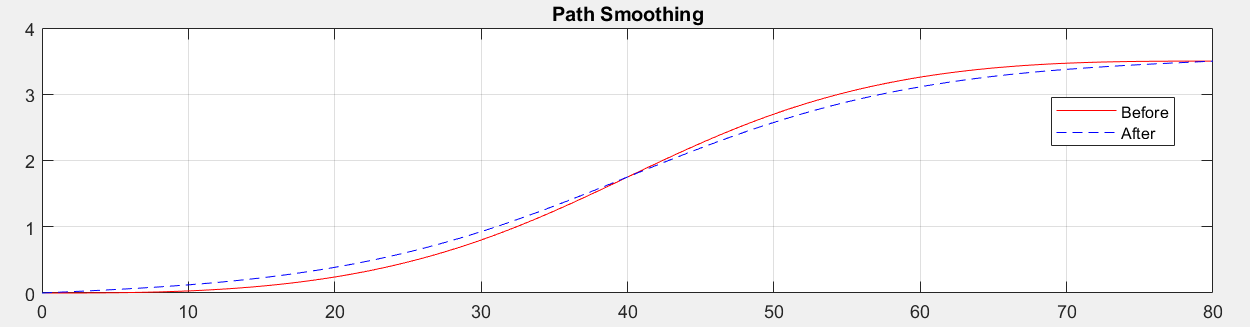
In this smoother, we penalized the norm of the offset between reference line and optimized trajectory, and the norm of . The objective function is as follows:

We use gradient descent method (do derivative over ) to minimize objective function by iterative search.

We also add some stopping criteria to ensure the robustness of the method. The first term is constrained by add a maximum offset between the reference line and optimized trajectory, one the offset increases upper to the upper bound, the iteration will be forced to stop. In terms of curvature, we add a maximum curvature constraint as well. In each iteration, we will check the curvature of the optimized trajectory, once all the points on the trajectory satisfy the curvature limit, iteration will also be stopped. To fix the running time of the iterative search part, we lastly constrained the maximum iteration number as a constant. Several examples of the smoother performance are as follows, where we are smoothing the lane changing reference line, with lateral offset 3.5m, and longitudinal variation between 50m to 80m.







As for the calculation time added by smoothing module, we use a simple example to show the approximate running time. In this example, we smooth an original Bezier curve with lateral offset of 3.5m, and longitudinal distance of 50m, the smoothing parameter alpha is set as 0.01, and beta 0.7, the running time is 62.135 *ms*, and the iteration loop number is 124.

## Kappa calculation

In our module, we provide two difference kappa calculation methods, one is deducted by pure math, while the other uses discrete interpolation.

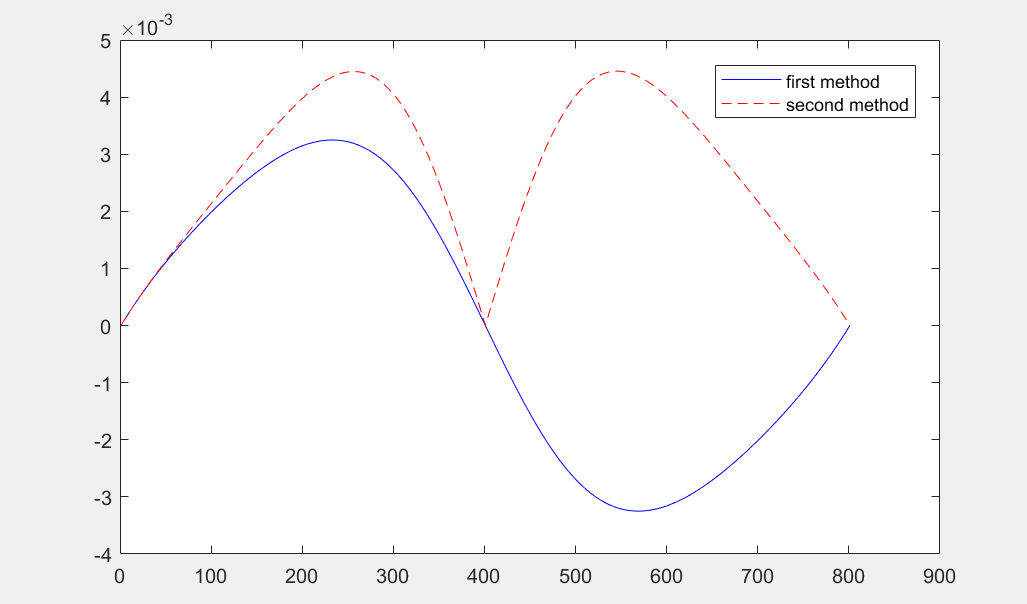
The first one use close form equation as follows:

The other method is as follows:

where

and

In the next few steps of path smoothing, we decided to trust the first method. And the two methods comparison is as follows:



# referemce

[ ] Werling M, Ziegler J, Kammel S, et al. Optimal trajectory generation for dynamic street scenarios in a frenet frame[C]//2010 IEEE International Conference on Robotics and Automation. IEEE, 2010: 987-993.