Self-Driving Car ND

Path Planning

1. Concept Of Operations

Before I describe the algorithm architecture and the data flow, we have no other choice but to describe the requirements that lead to the design.

1.1. Requirements

The requirements are:

- The car should try to go as close as possible to the 50 MPH speed limit
- The car should avoid hitting other cars at all cost as well as driving inside of the marked road lanes at all times, unless going from one lane to another.
- The car should be able to make one complete loop around the 6946m (4.31miles) highway
- The car should pass slower traffic when "possible" (note that other cars will try to change lanes too). The "possible" term means:
    - Without colliding
    - Without going outside the road
    - Without braking the speed limit
- The car should not experience total acceleration over 10 m/s^2
- The car should not experience a "jerk" that is greater than 50 m/s^3. Recommendation: it would probably be better to average total acceleration over 1 second and measure jerk from that.
- The simulator time step (dt) is fixed to 0.02 seconds.

Some additional notes:

- Our car will go as close as possible to the road max legal speed and try to pass when it is safe to do so or slow down if it is not possible.
- Left lane changes will be given priority if both, left and right lane changes, are possible maneuvers.
- Right lane changes to pass are permitted.

2. The Simulator (Sim)

One of the most important things to know about and that you don't find anywhere clearly described is "how the simulator works?" It's critical to understand what the Sim sends to us

and what expects from us to be able to successfully develop this Behavioral Path Planner. Not only that but most importantly what does the Sim do with what we give him

2.1. What does the Sim send to us?

The data sent to us by the Sim can be divided in 3 main groups:

2.1.1. Ego data

The Ego data consist of:

- Ego's X coordinate: This is the Euclidean X coordinate on the MAP reference system.
- Ego's Y coordinate: This is the Euclidean Y coordinate on the MAP reference system.
- Ego's s coordinate: This is the Frenet S coordinate on the MAP reference system.
- Ego's d coordinate: This is the Frenet s coordinate on the MAP reference system.
- Ego's Yaw coordinate: This is the Ego's heading angle on the MAP reference system. This refers to the direction where the longitudinal axes running from tail to front on the center line is pointing. Don't confuse this angle with the steering angle.
- Ego's Speed: This is the magnitude of the velocity vector and is independent from the coordinate system. As it is obvious, the speed is the same on both reference systems (if we consider them inertial systems). I'll make some assumptions later regarding s and d coordinates and the velocity magnitude.

2.1.2. Previous Path (left over)

These are 2 vectors containing the previous path X's and Y's that the Sim didn't cover at the time where it was sent to us. While this data and other data packages are being prepared to be sent to us, and while they are being sent and later processed by us, the SIM keeps running/moving the car to the next points (x,y) on a different thread. This means that we DON'T really know where Ego is (or will be) while we are processing the data. I will discuss this issue later in detail. For this reason we have to make a trade-off regarding what points we sent to the sim

2.1.3. Previous Path End "s" and "d"

These are simply the s and d Frenet coordinates of the previous path END point. At first glance you might be asking yourself why do I need this, or why does the Sim send these values back. Again, I will discuss this question as well as the issue above later.

2.1.4. Sensor Fusion

The sensor_fusion variable contains all the information about the cars on the right-hand side of the road, i.e. driving on the same direction as Ego and only on its vicinity.

The data format for each car is: [id, x, y, vx, vy, s, d]. The id is a unique identifier for that car. The x, y values are in global map coordinates, and the vx, vy values are the velocity components, also in reference to the global map. Finally s and d are the Frenet coordinates for that car.

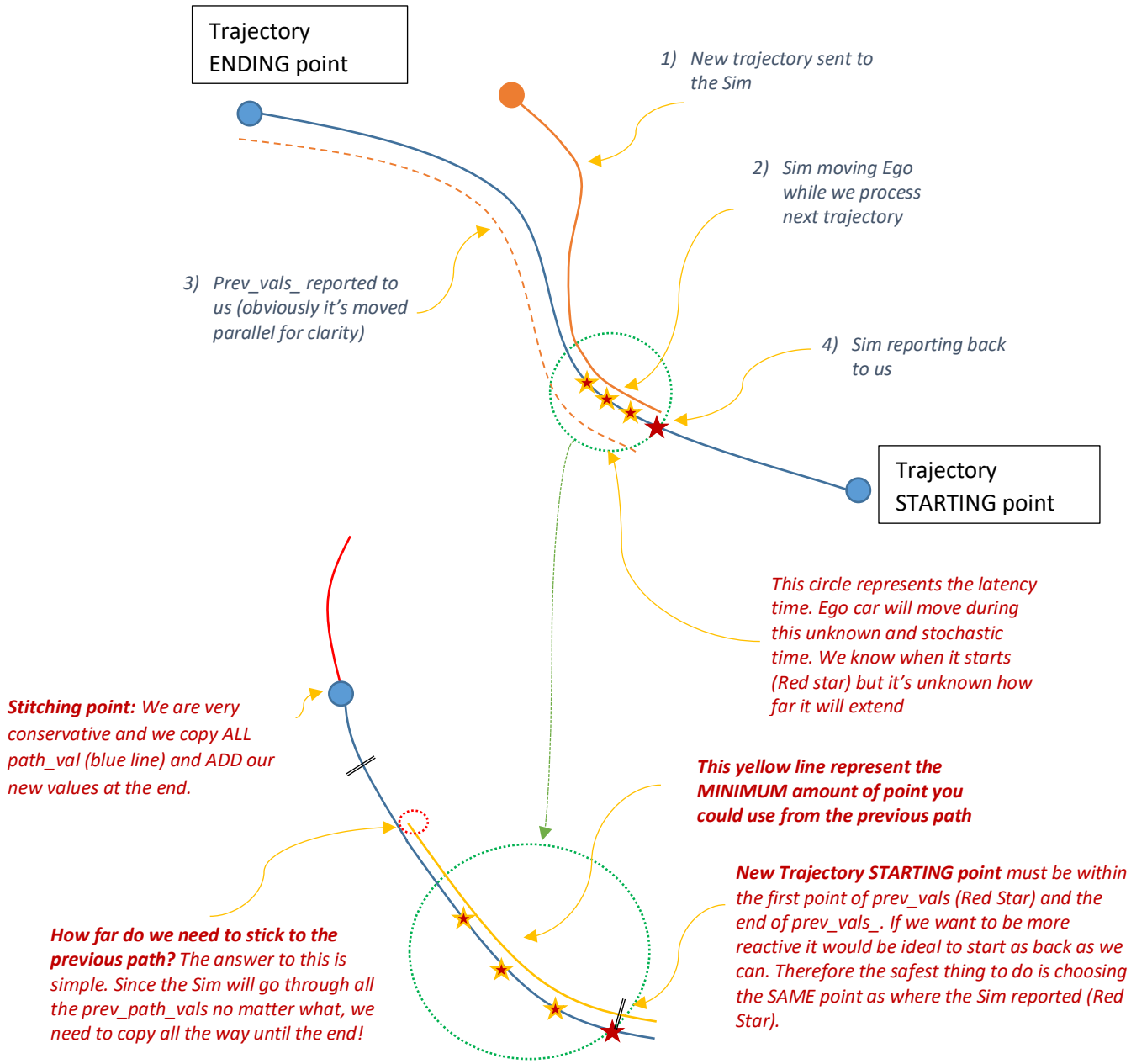
3. How does the Simulator work in this assignment?

One of the key elements on this assignment is to understand what does the Sim do with the set of (x,y) points we sent to it. Here are, in a compact way, the main points to take into account:

- Once you send a set of (x,y) to the Sim, remember that our Ego car will go through every single one of them, no matter what. What I'm trying to show here is that if we send 1000 pints (that is 20 secs worth of driving at 0.02sec time step), we are losing our capacity to react to any event that occurs during this period. For example, if during this time something changes in our environment such as a car moving in front of us and slowing down, or we have a car in front that the Sim didn't report because is more than 20 secs away from us, we will crash!
- On the other side of the spectrum, if we send just a few points (x,y) to the Sim trying to achieve a "semi"- real time behavior we will see that due to all the accumulated latencies, the Sim will run out of points very soon and the car will stop in the middle of the highway without knowing where to go next.
- This situation leads me to this final bullet point that clearly exposes the need to find a tradeoff between sending too many and therefore lose "control" over how fast we can react in a realistic highway environment and sending too few and have our car freeze waiting for the next wave of points and also breaking the laws of physics by moving and stopping instantaneously.

So where is this "sweet spot"? It all comes with a trial and error approach since it depends purely on how fast you can send back to the Sim your new points. Things like your CPU clock time, how efficient your code is, how many "cout" you have, etc. will determine what's a good average of the number of points you can send…or you can do as I did and make it dynamic. When I'm cruising I send less points to be able to react faster…and when the decision has been that a safe lane change is possible, then I send the entire lane change. We will go more into this later but just a teaser I'll tell you that not always this is a good approach.

Finally and answering to this section's header question, you must understand that the sim will simply ADD the new points EXACTLY at the beginning of the previous path sent to us. In other words, the new trajectory will be added exactly where the SIM took a snapshot to send back to us "prev_path_x and y" values. Let me show you this graphically.


Trajectory
ENDING point
1) New trajectory sent to the Sim
2) Sim moving Ego while we process next trajectory
3) Prev_vals_ reported to us (obviously it's moved parallel for clarity)
4) Sim reporting back to us
Trajectory
STARTING point
This circle represents the latency time. Ego car will move during this unknown and stochastic time. We know when it starts (Red star) but it's unknown how far it will extend
Stitching point: We are very conservative and we copy ALL path_val (blue line) and ADD our new values at the end.
This yellow line represent the MINIMUM amount of point you could use from the previous path
New Trajectory STARTING point must be within the first point of prev_vals (Red Star) and the end of prev_vals_. If we want to be more reactive it would be ideal to start as back as we can. Therefore the safest thing to do is choosing the SAME point as where the Sim reported (Red Star).
How far do we need to stick to the previous path? The answer to this is simple. Since the Sim will go through all the prev_path_vals no matter what, we need to copy all the way until the end!

Understanding the graph above represents critical point. Understanding that all the paths MUST be stitched smoothly (mostly focusing on transition speeds, accelerations and jerks) will determine a big percent of your success.

Let's discuss briefly the graph and how this affected my approach. If you pay attention you'll notice that:

1. You HAVE to always copy some points from the previous path to be able to make a smooth transition.
2. Where to start to copy and how many of this points you need to copy is the question.
3. As addressed on the graph, the starting point is no brainer. It must fall from the first point of prev_vals_ and the last point of prev_vals_ . As I mentioned above, we would like to be as close as we can to "real-time" and for this reason it's ideal to start copying from the first point on prev_vals.
4. Where to end the copy or how many points should we copy? As suggested on the graph we can actually slide visually an hypothetical "piece" of the previous trajectory and see that me MUST chose a point outside the circle since we don't want to start building a new path starting from where Ego has already passed..and make it go backwards (massive accelerations and jerks). So we conclude that we can get anywhere from where the green circle is until the end of prev_vals_. Due to the stochastic nature of the latencies, we can only predict by trial and error what would be a "safe" point it time. That would be a point with the worst case latency possible. The other choice is simply go all the way until the end of prev_vals_

There is pros and cons for each possibility. To allow a smooth transition we must know, not only the position but also its higher derivatives, namely, velocity and acceleration. This required knowledge is what's plays a huge role in deciding what approach to use.

If you decide to save the "last trajectory" (maybe as a function) you might be able to evaluate positions, velocities and accelerations at any given time "t" within the maneuver (trajectory) duration. The other method I tried was and finally sticked to it was too simple store the "status" (pos, vel, accl) of the last point sent to the Sim. That way I knew everything I needed to make a smooth transition.

4. Finite State Machine (FSM) Design

To comply with these requirements a finite state machine (FMS) can be used and implemented here to make decisions based on different driving scenarios.

My initial approach reached an architecture that I'll show below.

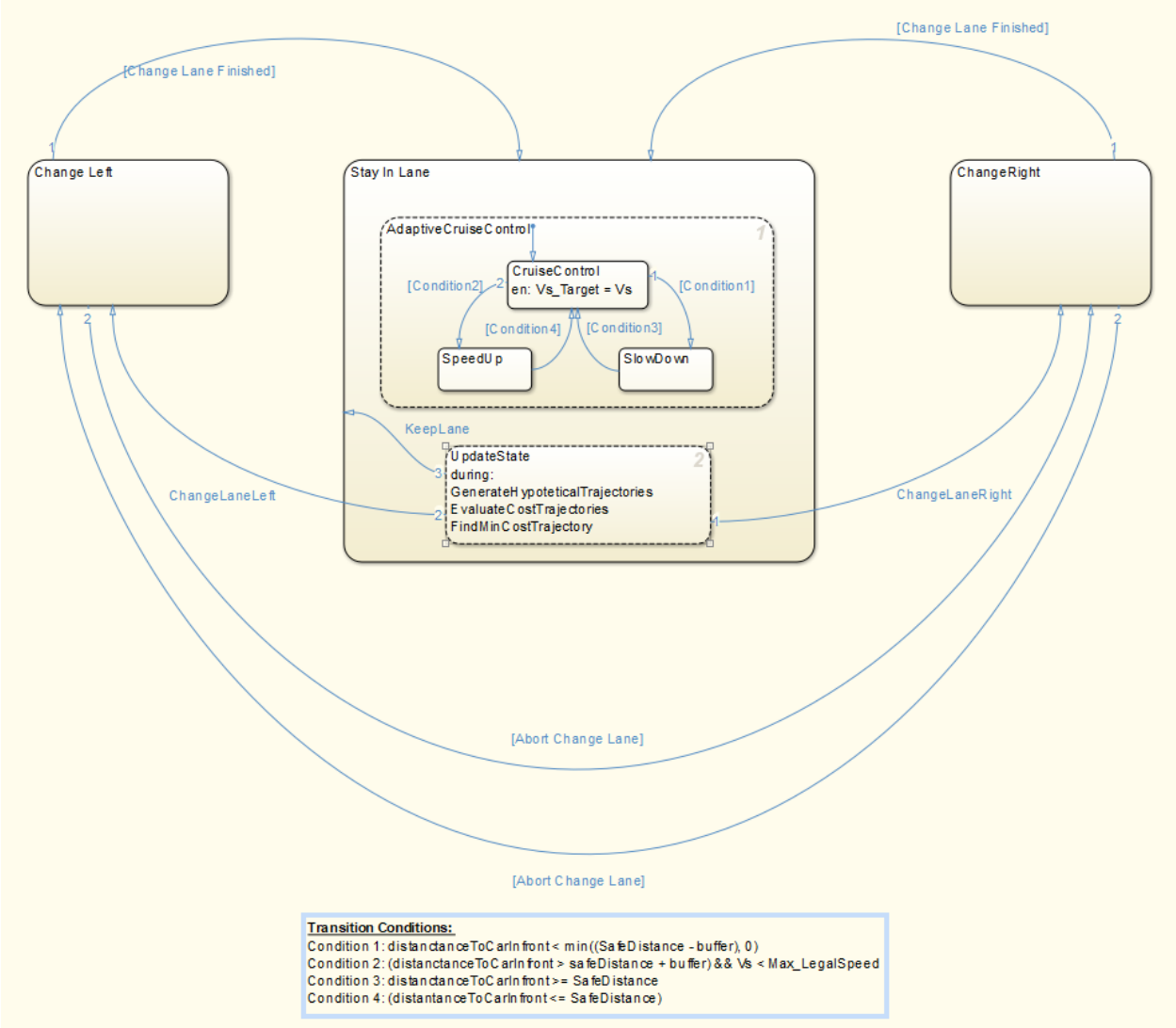


*Figure 1 Behavior Planner **Initial** Architecture*

This architecture was used for preliminary tests such as:

- Non-acceleration Lane changes with no obstacles (no car in front)
- Smooth transitions at different time steps dealing with process delays.

There are a few "criticizable" aspects of this initial design. Here are some, just to name a few:

- There are many scenarios where this architecture will fail. For instance, any motion of other cars in adjacent lanes is not considered appropriately and if we are changing lane (left or right) and the car in front of us decides to perform the same change lane, we will get in a dangerous situation that, according with our FSM, will make the Ego car abort the maneuver and try to get back, but this would actually make the situation worst since we will increase the changes to collide. Similarly, if we are in the "Stay In Lane" state and a car from an adjacent lane comes close to us, our FSM will likely choose to change lane to avoid a collision, but it's NOT likely that the Ego car will slow down and make room for the car on the adjacent lane that want to merge in our lane. All these different scenarios tend to guide us in creating a very complex state machine that will probably end up being "too" specific and lose the capacity to generalize its safe behavior.
- Taking the "Update State" (or "Decision Making" as I renamed later) outside the "Stay in Lane" state (as it is on the lecture files) might solve some issues since we will be continuously assessing the state of the vehicle. But, what would be the decision on the scenarios presented above? When you are in the middle of the change lane maneuver and you exactly in the middle of this event, a transition to "Keep Lane" would not make too much sense: does it mean going back or finishing the maneuver?
- Although this state machine is relatively simple, it is computationally heavy since we have to predict the trajectories (collection of [s,d] for a "Horizon/dt" number of samples) of all "relevant" vehicles on the road and after check for "collision" and "safe distance" with our Ego car. In a safety critical application like this one, computational efficiency plays a big roll and I was wondering if we could have a "Decision Making" module that did NOT need to loop over all vehicles and all time steps for the duration of the horizon to make a safe decision.
- The point above unveiled the lack of formalism to distinguish a-priori (before predicting all trajectories and checking for unsafe conditions) from a safe and unsafe lane change maneuver.
- The "SafeDistance" parameter used in this initial approach, (refer to the transitions on figure 1 above) is a dynamically calculated distance that depends on the Ego car speed and that will allow the Ego car to stop and not collide with the car in front if the latter was to come to a full stop (assuming we apply in both the max decel constraint above). The use of this parameter, although sufficient for this assignment, was a "one-solution fit all" type of approach.

After a deep dive on the literature to investigate how researchers and the industry tackle these issues, I narrowed down my concerns to the following:

- What are the minimum number of parameters/indicators (based on the data we get from the Simulator: both velocity and position of ego vehicle and surrounding vehicles) that can be used to "decide" our Ego Car behavior if we wanted to do a one-time calculation without looping over all vehicles and predictions?
- What are the minimum number of cases to consider that can cover the majority of situations that we might encounter on the simulator? i.e. what's the smallest number of states that I need to identify to cover most if not all cases in highway driving?

Well, after a broad research over many publications, Master and PhD thesis I discover the following.
Two of the main types of highway-driving potential dangers are:

- Driving with very low inter-vehicle distance and
- Driving with high speed differences.

The inter-vehicular indicator is used for the case where two vehicles are driving too close to each other while the Time-To-Collision indicator is used to for two vehicles having big speed differences. Both of the indicators are used in the "behavior planning" module to decide whether it is safe to make a lane change or whether the final predicted vehicle position is safe after lane change has been conducted.

Time to Collision (TTC)

The first indicator is the time to collision indicator and it is defined as the time required for two vehicles to collide if both vehicles continue to drive at their current speeds and, of course, the direction or heading angle of the vehicle is also kept as it is. TTC can be calculated as:

$$\mathrm{TTC} = \frac{x_f - x_r}{\mathrm{v_r} - \mathrm{v_f}}$$

Equation 1

Where the numerator shows the relative distance between two vehicles (xf is the front or receding vehicle's position and xr is the rear or following vehicle's position) and the denominator is the relative velocity (vf is the velocity of the leading vehicle and vr is the velocity of the following vehicle). Based on projects like ARCOS (www.arcos2004.fr) and PREVENT www.preventip.org), several thresholds are defined to indicate the potential dangerous based on the calculated value of TTC as follows:

1. TTC = 10s, two vehicles are suggested to be safe and not interacting.
2. TTC = 1.5s, used as the first level warning.
3. TTC < 1.3s, used as the second level warning.
4. TTC < 1s, automated force brake control is suggested to be applied.

This means that when TTC goes below 1s, it is highly possible for a collision to occur and this is also the same time for a driver to make reaction. In our case, I will choose a conservative value, let us say TTC = 1.5s, used as the first level warning.

Inter-Vehicular Time (TIV)

The second indicator is the inter-vehicular time indicator. We need this indicator as well because TTC can only consider the case where two vehicles have relatively big velocity difference (and therefore the denominator is not close to zero). But if two vehicles are driving at the same speed, and driving at extremely close distance, this is still considered highly dangerous while TTC will be left in a singularity that usually will be solved giving a 0. To also take this kind of cases into consideration, the indicator TIV is coming into play. TIV is defined as follows,

$$\mathrm{TIV} = \frac{xf - xr}{\mathrm{vf}}$$

Equation 2

Where the symbols used have the same meaning as described on the TTC.
With these two indicators explained, we can jump now on to our second question regarding the minimum number of cases that we should consider in our design that will encompass most, if not all, the real possible cases. The ultimate objective I am pursuing by answering this question is to identify the states that the Finite State Machine should have to encompass a complete highway-driving scenario. In my initial approach, as well as on the early lectures, we identified 3 states: Keep Lane, Change Lane Left and Change Lane Right. We know this was an oversimplified approach that could possibly pass the requirements for this assignment, but with a huge uncertainty if it will "always" succeed at it.
Let us present the most generic scenario and try to address all possible cases.

Highway Generic Scenario

Let's consider a "Vehicle A", always in front of the Ego car (Vehicle E) and traveling always slower than the Ego Car. Let's also consider a third vehicle, "Vehicle B" on an adjacent lane to the previous two. Let's also define L_b and L_E as the lengths of vehicles "B" and "E" respectively.

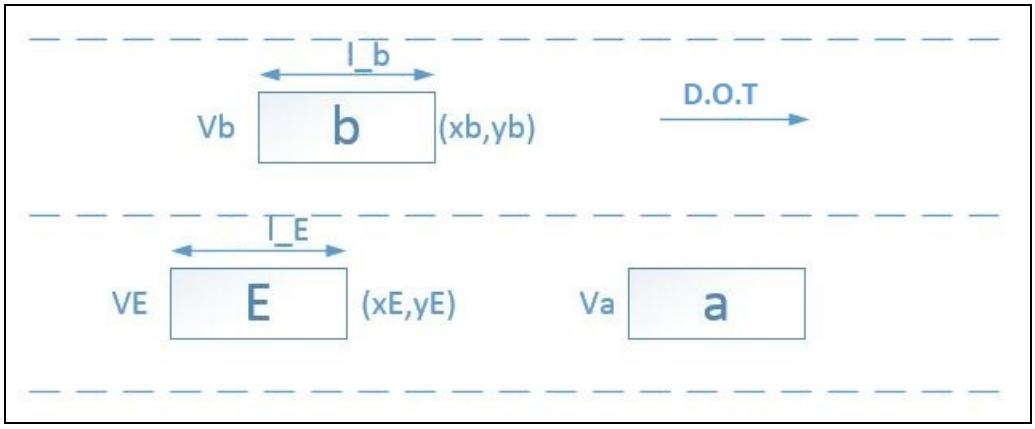


Figure 2 Highway Generic Scenario

In this generic scenario, we can foresee that the 2 factors to take into account are:

- The relative positions between our Ego car and Vehicle B, creating 3 possibilities shown below

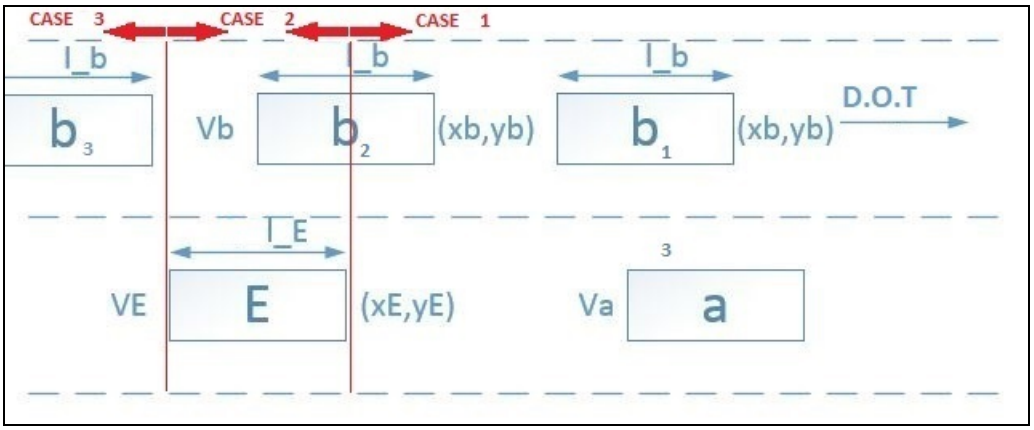


Figure 3 Relative position of Vehicle B and Ego car

- The relative Speeds between our Ego car, Vehicle B, Vehicle A and our Desired speed ($\boldsymbol{V_{des}}$) (that not necessarily has to be the Speed Limit as it is in this assignment).
    1. $\boldsymbol{V_a < V_{des} < V_E < V_b}$
    2. $\boldsymbol{V_a < V_{des} < V_b < V_E}$
    3. $\boldsymbol{V_a < V_b < V_E < V_{des}}$
    4. $\boldsymbol{V_a < V_E < V_b < V_{des}}$
    5. $\boldsymbol{V_b < V_a < V_E < V_{des}}$

Combining these cases, we should get 15 different scenarios that will cover all major driving situations. To simplify this cases, let's assume that our Desired speed ($\boldsymbol{V_{des}}$) is the Speed Limit and can NOT be violated. In this case, we will end up with:

1. $\boldsymbol{V_a < V_{des} = V_E < V_b}$
2. $\boldsymbol{V_a < V_b < V_E < V_{des}}$
3. $\boldsymbol{V_a < V_E < V_b < V_{des}}$
4. $\boldsymbol{V_b < V_a < V_E < V_{des}}$

To compartmentalize these cases and make it easy to follow, let's simply make 2 mayor divisions and later look at subcases.

Faster vehicle on the adjacent lane ($\boldsymbol{V_E < V_b}$)

- **Case 1: $\boldsymbol{V_a < V_{des} < V_b}$**

  In this case, we can see that our Ego car will have to pass Vehicle A, but it will have to make some interesting decisions to figure out if it can pass ahead of B or behind B. This precise question brings up the following subcases:

    - **Case 1.1: B is ahead of Ego:**

      $$\boldsymbol{x_b > x_E + 0.5\, L_E + 0.5 L_b}$$

      This is the case where vehicle b has the fastest velocity and already driving longitudinally in front of the Ego vehicle. In this case, a normal optimized lane

change trajectory should be initiated i.e. the trajectory designed for the case where there is no obstacle. We will call this state **Normal lane change** (with no acceleration). We do not want to speed up at Vehicle B speed since it's already higher than the speed limit

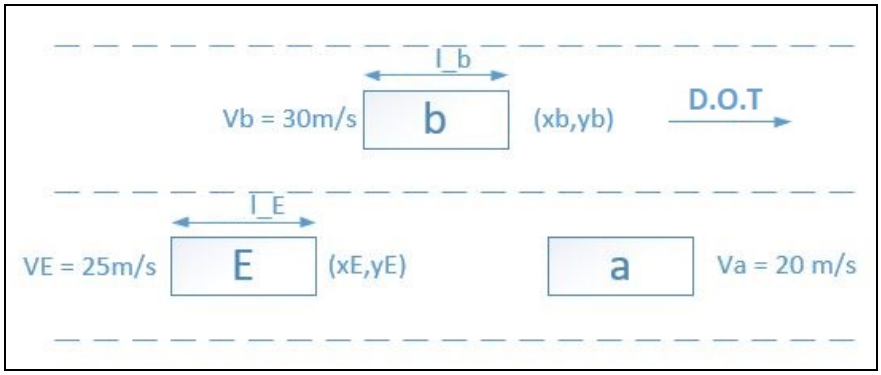


Figure 4 Case 1.1

- **Case 1.2: B is "Slightly" behind Ego:**

$$(x_E - 0.5\,L_E - 0.5L_b) \leq x_b \leq (x_E + 0.5\,L_E + 0.5L_b)$$

In this case, since vehicle b is driving at a higher speed than the ego vehicle, and is behind the ego vehicle, ego should wait until vehicle b passes. We will call this state **Adaptive Cruise Control**. In this state, as in my initial State Machine, we will slow-down and maintain a safe distance with vehicle A until B passes. After that we should be in under Case 1.1 where B is already

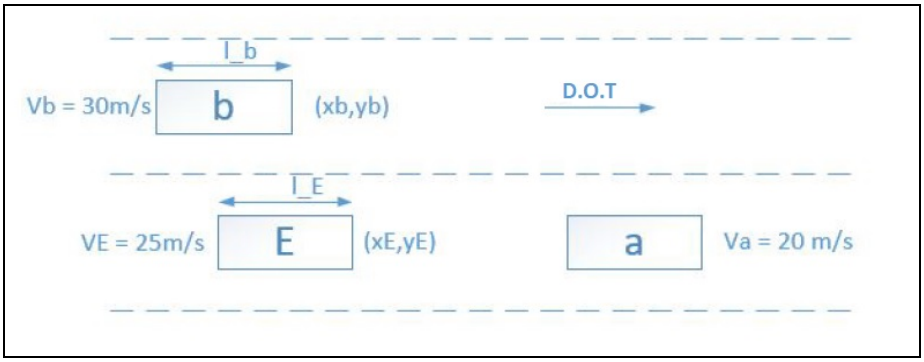


Figure 5 Case 1.2

- **Case 1.3: B is "Far" behind Ego:**

$$x_b < (x_E - 0.5\,L_E - 0.5L_b)$$

When vehicle b is behind the ego vehicle, the longitudinal relative distance come into play. In order to execute a "safe" lane change while vehicle B is driving at a higher speed than the ego vehicle, both TTC and TIV, as discussed above, are used as a decision making criteria. In this case, if the relative distance is small enough based on TTC and TIV for a lane change, then a lane change command can be given and the trajectory generator will try to generate a safe trajectory. The speed limit

V_{Limit} is also added because vehicle b even though is going faster than the desired velocity, it could actually be lower than the speed limit. This was done to cover the more general case where the "Desired Speed" could be lower than the Speed Limit. In these cases, the Ego car may accept accelerate to a certain higher speed in order to pass around the slow preceding vehicle, but the velocity cannot exceed the road Speed limit. This state is called **Acceleration Lane change** with a target Ego speed at the end of the maneuver equal (or higher if is under the speed limit) to the speed of vehicle B. $\boldsymbol{V_{E\_final} = V_b}$ (Leading vehicle b at a safe distance)

The requirements on TTC and TIV to trigger/transfer to this state are:

$$\text{TTC} = \frac{x_E - x_b - 0.5L_e - 0.5L_b}{v_b - v_E} > 1\ sec$$

$$\text{TIV} = \frac{x_E - x_b - 0.5L_e - 0.5L_b}{v_b} > 0.5\ sec$$

$$V_b - V_a < V_{Limit} \quad Max\ Speed\ (Speed\ Limit)$$

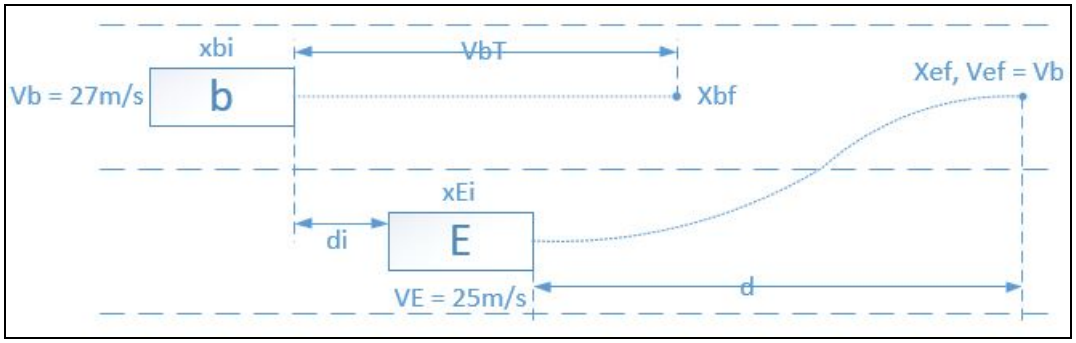


Figure 6 Case 1.3

In this scenario, it is evident that we first need to calculate T using the approximated optimized solution to we can find out d, that it should be used by the trajectory generator and smoother modules.

- **Case 1.4: B is in a position such as it doesn't comply with any of our TTC and TIV:**

$$\text{TTC} = \frac{x_E - x_b - 0.5L_e - 0.5L_b}{v_b - v_E} < 1\ sec$$

OR

$$\text{TIV} = \frac{x_E - x_b - 0.5L_e - 0.5L_b}{v_b} < 0.5\ sec$$

OR

$$V_b - V_a > V_{Limit} \quad Max\ Speed\ (Speed\ Limit)$$

This case is very similar to case 1.3 with the only difference that in this case, vehicle B is coming at very high speed relative to our Ego car (and vehicle A in

front) and therefore we will have to wait and let it pass before we attempt our change of lane maneuver. Under these conditions, we will actually go to the same state as in case 1.2: **Adaptive Cruise Control**. As explained before, we will slow-down and maintain a safe distance with vehicle A until B passes.

- **Case 2: $V_a < V_E < V_b < V_{des} \sim SpeeLimit$**

  In the same fashion as before, this general case brings up a few question that our new "Decision Making" state machine should be able to handle. For instance, depending on Vehicle B position and relative speed, we might have to wait, like we did before, even though is going at a slower speed than our desired speed but faster than us. We should be able to find that "critical" point where we can no longer pass ahead of it while staying in our safe bubble. To address this multiple possibilities and find this threshold, let's build up all the subcases as we did before.

  - **Case 2.1: B is ahead of Ego:**

    $$x_b > x_E + 0.5\,L_E + 0.5L_b$$

    Here, this is the case very similar to Case 1.1. However, the difference is the ego vehicle wants to drive at a speed faster than vehicle b. In this case, and since we want to travel faster, we can make an acceleration lane change but with the final velocity, at least equal if not trying to reach the Speed limit, to the vehicle b and following behind it. This state is, as in case 1.3, an **Acceleration Lane change** with a target Ego speed at the end of the maneuver equal to Vehicle B speed or the Speed Limit.

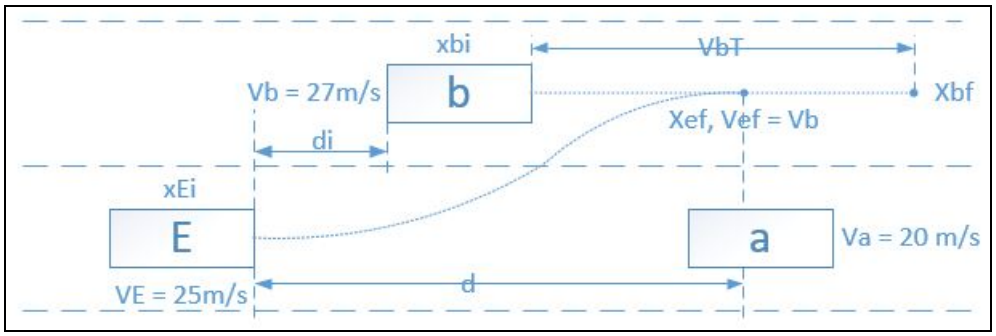


Figure 7 Case 2.1

Since an acceleration lane change need to be made, both d and T again need to be calculated. At the end of the maneuver, TIV indicates the following must be hold,

$$\text{TIV} = \frac{x_{b\_final} - x_{E\_final} - 0.5L_e - 0.5L_b}{\text{V}_{\text{E\_final}}} > 0.5\ sec$$

  - **Case 2.2: B is "Slightly" behind Ego:**

    $$(x_E - 0.5\,L_E - 0.5L_b) \leq x_b \leq (x_E + 0.5\,L_E + 0.5L_b)$$

    In the same fashion as in case 1.2, and since the velocity of vehicle b is higher, ego vehicle should just choose to wait until vehicle b pass. Then, it enters into Case 2.1 where an Acceleration Lane Change maneuver will be initiated.

- **Case 2.3: B is "Far" behind Ego:**

$$\boldsymbol{x_b < (x_E - 0.5\, L_E - 0.5 L_b)}$$

And

$$\text{TTC} = \frac{x_E - x_b - 0.5L_e - 0.5L_b}{\text{v}_\text{b} - \text{v}_\text{E}} > 1\ sec$$

$$\text{TIV} = \frac{x_E - x_b - 0.5L_e - 0.5L_b}{\text{v}_\text{b}} > 0.5\ sec$$

$$\text{V}_\text{b} - \text{V}_\text{a} < \text{V}_\text{Limit} \quad Max\ Speed\ (Speed\ Limit)$$

This is very similar to Case 1.3 but since vehicle b is driving at a slower velocity than the desired velocity (Speed Limit), an **Acceleration Lane change** maneuver should be initiated with the final velocity of the ego vehicle equal to the desired velocity. Acceleration Lane Change with $\boldsymbol{V_{E\_final} = V_{des}}$ (Leading vehicle b at a safe distance)

- **Case 2.4: B is in a position such as it doesn't comply with any of our TTC and TIV:**

$$\text{TTC} = \frac{x_E - x_b - 0.5L_e - 0.5L_b}{\text{v}_\text{b} - \text{v}_\text{E}} < 1\ sec$$

OR

$$\text{TIV} = \frac{x_E - x_b - 0.5L_e - 0.5L_b}{\text{v}_\text{b}} < 0.5\ sec$$

OR

$$\text{V}_\text{b} - \text{V}_\text{a} > \text{V}_\text{Limit} \quad Max\ Speed\ (Speed\ Limit)$$

The other relative positions under this velocity will need to wait until one of the other event is triggered. Then, corresponding lane change maneuver can be generated.

Slower Vehicle in the Adjacent Lane ($\boldsymbol{V_E > V_b}$)

Case 1: $\boldsymbol{V_b < V_a < V_E < V_{des} \sim SpeeLimit}$

This is the case where Vehicle B is now the slowest of all three vehicles.

- **Case 1.1: Vehicle b is leading the Ego: $\boldsymbol{x_E \leq x_b}$**
  Since the vehicle b is driving at the lowest speed among the three vehicles, if the vehicle b is leading the ego vehicle, then the ego vehicle can just stay in its lane behind A until it passes B. After, it can try to pass A in front of B (Case 1.2) below.
- **Case 1.2: Ego is leading the Vehicle b: $\boldsymbol{x_E > x_b}$**
  Now the vehicle b is surpassed by the ego vehicle or, in other words, the ego vehicle is leading, then a **Normal lane change** (with no acceleration) can be commanded. There is no need to accelerate since b is slower than Ego. As I

mentioned above, we can always substitute this for an **Acceleration Lane change** with a target Ego speed at the end of the maneuver equal to the speed limit.

- **Case 2:** $V_a < V_b < V_E < V_{des} \sim SpeeLimit$
  Although both vehicle a and vehicle b are driving at a slower velocity than the desired velocity, the adjacent lane i.e. vehicle b is still faster than the current lane and a lane change will be commanded depending on vehicle B position.
  - **Case 2.1: Vehicle b is ahead of Ego:**

$$x_b > x_E + 0.5\,L_E + 0.5L_b$$

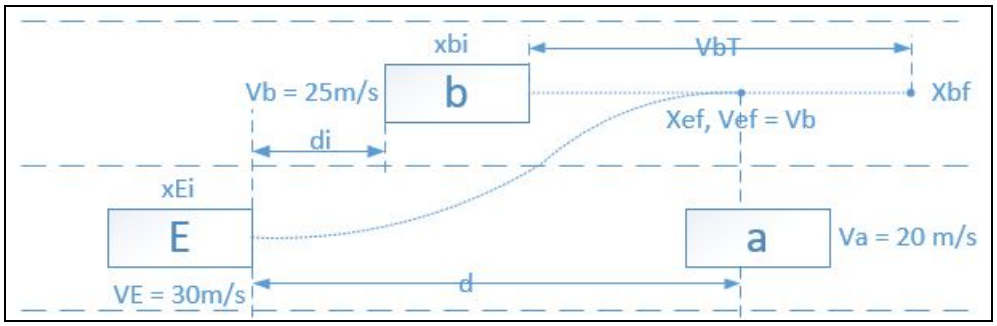


Figure 8 Case 2.1 (Slower Vehicle in adjacent lane)

As in the Case 1.3 above, trajectory parameters need to be calculated for this event. TIV > 0.5 holds.

  - **Case 2.2: B is "Slightly" behind Ego:**

$$(x_E - 0.5\,L_E - 0.5L_b) \leq x_b \leq (x_E + 0.5\,L_E + 0.5L_b)$$

In this case, the decision making can be depending on the type of driver. Some drivers might want to have an **Acceleration Lane change** trajectory while some drivers prefer to wait until the vehicle b pass and then do the lane change. Here, to be more conservative and ensuring safety, let's make our system recommend wait and then make the lane change.

  - **Case 2.3: B is "Far" behind Ego:**

$$x_b < (x_E - 0.5\,L_E - 0.5L_b)$$

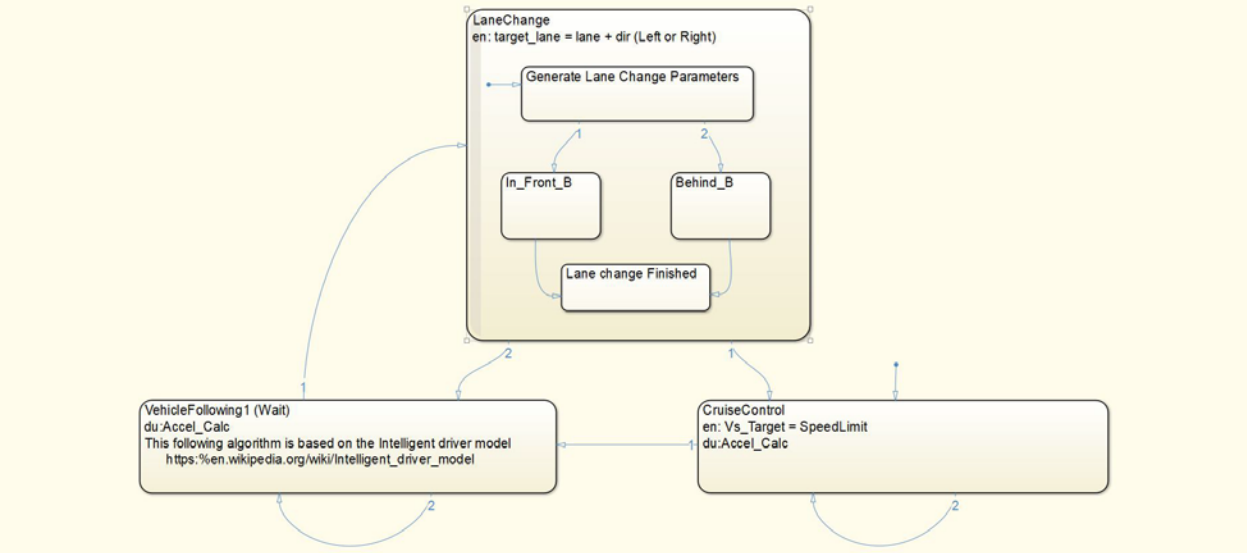
In this case, since vehicle is already behind the ego vehicle and driving at a slower velocity than the ego vehicle, a lane change maneuver should be initiated. Moreover, because currently, ego vehicle is slower than the desired velocity, an **Acceleration Lane change** is desired with an Ego final speed around the Speed limit.

After describing all the possible scenarios, the states (maneuvers) and the conditions that will be used to trigger them (transition conditions) we can present them.

State (Maneuver)	Description

Adaptive Cruise Control (Vehicle Following)	Stay in Lane and adapt speed to a safety speed. When possible (TTC and TIV safe conditions) speed up to Speed Limit.
Acceleration Lane change (Left or Right)	Ego car will accelerate maintaining Acceleration max requirements and Jerk max requirements and change lane. To perform this maneuver we will need: • Target Lane (d Frenet). Only allowed 1 change of lane at the time. • Distance "d_safe" that will help create the safe trajectory. • Time (T) that the maneuver/trajectory should last. • Target final Ego car Speed.
Non-Acceleration Lane change (Left or Right)	Simple lane change at constant speed with no obstacles (no d_safe or hard time constraints)

5. Final State Machine Architecture



6. Algorithm flow

The steps taken are:

1. Even though we have the current Ego's data, we know that we will not be able to act until the end of our previous trajectory. For this reason, the first step I did was to "Move" all the vehicles on the road an equivalent amount of time left on my previous path. I want to know where all vehicles will be when I can add my new trajectory.

2. Generate all the possible maneuver based on where Ego is right now (for emergencies) but mainly based on where Ego and the rest of the Vehicles will be at the end of the previous trajectory.
3. Calculate the “best” trajectory using:
    - Jerk Minimization Trajectory approach (with added perturbed goals to explore possible better trajectories)
    - And a “Minimization Cost” approach to select the “best” one.
4. Once we have the best trajectory, we just need to make a smooth transition from the previous trajectory.
5. Save the status (pos, vel, accel) of last point of this new trajectory
6. Send the new smooth trajectory to the Sim.