reflection on how to generate paths.

The code for generating optimal paths is located in lines 262-391 of the code.

In lines 262-264 we declare 3 vectors:

- **gap_ahead** contains the distance between the ego vehicle and the nearest car ahead in each lane
- **gap_behind** contains the distance between the ego vehicle and the nearest car behind in each lane
- **speed_ahead** contains the speed of the nearest car ahead in each lane ahead of the ego vehicle

The index of the vector values corresponds to the lane where the vehicle is driving.

```
262             vector<double> gap_ahead(3); //vector containing the distance between the ego car and the nearest vehicle ahead for each lane
263             vector<double> gap_behind(3); //vector containing the distance between the ego car and the nearest vehicle behind for each lane
264             vector<double> speed_ahead(3); //speed of the cars ahead of us
```

In lines 275-330 we loop through the sensor fusion data. The index i in the for loop in line 275 refers to a specific car that is sensed by the ego car's sensors. In lines 290-301 we check in witch lane vehicle i is driving.

```
275             for (int i = 0; i < sensor_fusion.size(); i++)
276             {
277             //car is in my lane
278             double vx = sensor_fusion[i][3];
279             double vy = sensor_fusion[i][4];
280             double check_speed = sqrt(vx*vx + vy*vy);
281             double check_car_s = sensor_fusion[i][5];
282
283             float d = sensor_fusion[i][6];
284             //check the lane of the current vehicle
285             int this_lane;
286             check_car_s += ((double)prev_size*.02*check_speed);
287             //std::cout << "speed_ahead[0]: "<< speed_ahead[0] << "speed_ahead[1]: " << speed_ahead[1]<< "speed_ahead[2]: " << speed_ahead[2]<<endl;
288
289             //check the lane of the current sensor fusion car
290                 if (d < 4 && d > 0)
291                 {
292                     this_lane = 0;
293                 }
294                 else if (d < 8 && d > 4)
295                 {
296                     this_lane = 1;
297                 }
298                 else
299                 {
300                     this_lane = 2;
301                 }
```

In lines 303-317 we check if car i is ahead or behind the ego vehicle and enter the distance and speed in the gap_ahead and speed_ahead vectors if car i is the nearest vehicle in its lane.

```
303             if (check_car_s > car_s)
304             {
305                 if (gap_ahead[this_lane] > check_car_s - car_s)
306                 {
307                     gap_ahead[this_lane] = check_car_s - car_s;
308                     speed_ahead[this_lane] = check_speed;
309                 }
310             }
311             else
312             {
313                 if (gap_behind[this_lane] > car_s - check_car_s)
314                 {
315                     gap_behind[this_lane] = car_s - check_car_s;
316                 }
317             }
```

In line 320-330 we check if there is a vehicle too close in front of the ego car and in the same lane. If this is the case we set the variable **too_close** to True.

```
319         //check distance between ego car and car in same lane ahead
320         if(d < (2+4*lane+2) && d > (2+4*lane-2))
321             {
322             //check_car_s += ((double)prev_size*.02*check_speed);//if using previous points can project s value outwards in time
323             double gap = check_car_s - car_s;
324             //check s values greater than mine and s gap
325             if((check_car_s > car_s) && ((gap) < 30))
326                 {
327                 too_close = true;
328                 }
329             }
```

In lines 332-387 we code the logic to find the optimal lane switching strategy. I used the following main principles to decide when to switch and to witch lane to switch:

- Only consider switching lanes if there is a slower car in the same lane ahead of you.
- Only switch if there is sufficient room ahead and behind the vehicle on the target lane.
- Only switch to a new lane if the car ahead of the ego car in the target lane is moving faster than the car ahead of the ego car in the current lane.
- If the ego car is in the center lane, then give preference to the lane where there is no car picked up by the sensor fusion data. This is the case when for this lane the value for speed_ahead[i] is zero. This is likely to be the least busy lane.

```
332             if (too_close)
333             {
334                 ref_vel -= .224;
335                 double mingap_ahead = 10*car_speed/49.5;//minimum distance required to change lane safely
336                 double mingap_behind = 20 * 49.5 / car_speed;;
337                 //left lane
```

In line 334 we start slowing down the ego vehicle if it gets too close to another vehicle ahead. In lines 335-336 we define the minim distance ahead and behind that is required for safe lane switching. This distance is a function of the speed of our ego vehicle. If the speed of the ego vehicle is high, then the required distance to the car ahead in the target lane is higher than in the case of slow speed. This is because of the relative speed between the ego car and the car in the target lane is high. The opposite is true for the required distance to the car behind the ego vehicle.

```
337             //left lane
338             if (lane == 0)
339             {
340                 if (gap_behind[1] > mingap_behind && gap_ahead[1] > mingap_ahead)
341                 {
342                     if (speed_ahead[1] > speed_ahead[lane] || speed_ahead[1]==0)
343                     {
344                         lane = lane + 1;
345                     }
346                 }
347 
348             }
```

If the ego car is in the left lane, then it can only chose between staying in the current lane or move 1 lane to the right. In line 340 we check if the required distances ahead and behind the ego vehicle are available to safely switch to the middle lane. If that is the case, we also check in line 342 if the car ahead of us in the middle lane is moving faster than the car ahead of us in the left lane. If this is true, it makes sense to switch to the middle lane. The same logic applies for the lane switching from the right lane to the middle lane.

```
349             //middle lane
350             else if (lane == 1)
351             {
352                 if (gap_behind[0] > mingap_behind && gap_ahead[0] > mingap_ahead && speed_ahead[0] == 0)
353                 {
354                     lane = lane - 1;
355                 }
356                 else if (gap_behind[2] > mingap_behind && gap_ahead[2] > mingap_ahead && speed_ahead[2] == 0)
357                 {
358                     lane = lane + 1;
359                 }
360                 else if (gap_ahead[0] > gap_ahead[2] && gap_behind[0] > mingap_behind && gap_ahead[0] > mingap_ahead)
361                 {
362                     if (speed_ahead[0] > speed_ahead[lane] || gap_ahead[0] > 60)
363                     {
364                         lane = lane - 1;
365                     }
366                 }
367                 else if (gap_ahead[2] > gap_ahead[0] && gap_behind[2] > mingap_behind && gap_ahead[2] > mingap_ahead)
368                 {
369                     if (speed_ahead[2] > speed_ahead[lane] || gap_ahead[2]>60)
370                     {
371                         lane = lane + 1;
372                     }
373                 }
374             }
```

If the ego car is driving in the middle lane, then it can either stay in this lane or move left or right. In line 352, we check if there is sufficient space to switch to the left lane. We also check if the left lane is traffic free (this is the case when the care is not sensing any cars in that lane and hence the speed of the car ahead is 0).

If this is not the case, then we check the same for the right lane (line 356).

If we don't find a safe lane without traffic, then we check in line 360 if the left lane is safe and has the most space ahead of us. I assume that the line with the most space ahead is the fastest and thus the best switching option. If this is the case, then I finally check in line 362 if the left lane is either the fastest lane, or there is sufficient space to accelerate and pass the slower vehicle in front of me.

```
388             else if (ref_vel < 49.5)
389             {
390                 ref_vel += .224;
391             }
```

Finally in lines 388-391 we let the vehicle accelerate to the maximum speed when there is no vehicle in front of our car with a distance of less than 30 meters.

With this lane switching algorithm the car drove 18 miles without an incident. Eventually it had an “outside lane incident” because it ended up in slow moving traffic with cars accelerating and braking all the time, which caused the ego car to swerve between 2 lanes.