Project P1 Term 3 – Model Documentation

Outline

This document describes the path planning model coded inside *main.cpp* file to successfully run the project 1 of term 3.

Model

The path planning model reuses the code architecture and trajectory module as discussed in the project Q&A plus new modules for lane change decision and velocity adaption.

Code Architecture

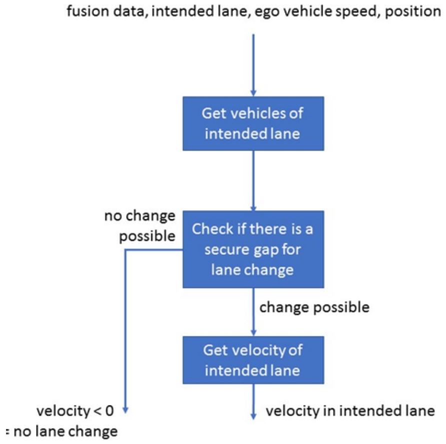
The model architecture mainly starts at lines 361. The code of the lines 361-404 is reused the from the Q&A starting to analyze the fusion data for vehicles inside the same lane as the ego vehicle. If a vehicle in-front of the ego vehicle is too close which is defined by *REF_DISTANCE*, the *too_close* variable is set triggering the **lane change analysis & operation** (if analysis is ok) or an **emergency break**. Both activities are new modules.

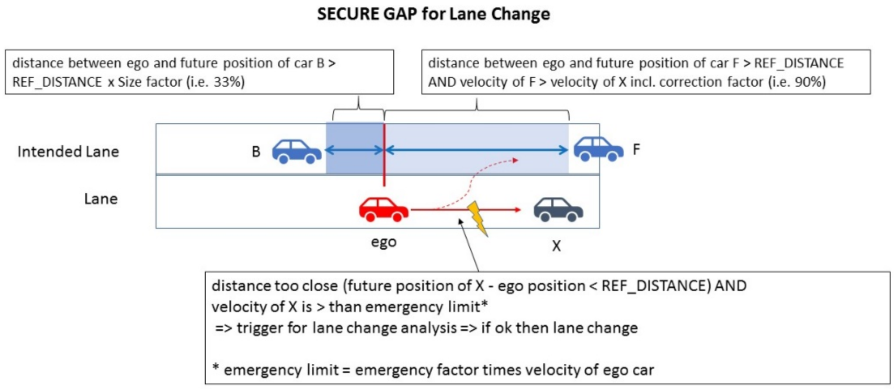
- **emergency break** is analyzed by checking the velocity of the vehicle in front. If it is smaller than a percentage of the ego velocity – the percentage is set by the *VELOCITY_EMERGENCY* define (i.e. 30%) - then no lane change analysis & operation will be performed and the velocity will be dramatically reduced – see velocity adaption module at line 464.

```
406    // -----------------------------------------------------------------------------
407    // check for lane change or emergency break
408    // -----------------------------------------------------------------------------
409
410    if (car_speed*VELOCITY_EMERGENCY > check_speed) emergency_break = true; // emergency break
411    else {
412
413    // lane change analysis
414
415    double speed_lane   = -1.0;
416    double speed_lane_l = -1.0;
417
418    // -----------------------------------------------------------------------------
419    // lane analysis by checking gap for lane change & speed at new lane
420    // -----------------------------------------------------------------------------
```

- the **lane change analysis** is triggered based on the *too_close* flag and if there was no emergency break. Based on the lane position of the ego vehicle, possible lane changes will be analyzed. A lane change is possible if it can be done without collision by defining a secure change gap and if the velocity of vehicles inside the intended lane are higher or equal than on the current ego velocity. Note: the “comparison” velocity of the ego vehicle can be decreased or increased by the *VELOCITY_DEC* which is multiplied with ego velocity i.e. multiplied by 0.95 for 95% of ego speed).
  This lane check analysis is realized by the *check_lane()* function specified in the lines 193-277. This function takes the fusion, lane and ego vehicle data and returns the minimum speed of vehicles inside the intended lane if the lane change is possible else it returns a negative value.

The figures below explains the change gap model for a lane change and the state model of the *check_lane()* function.





The lines 421-459 present to lane change analysis and in case of possible lane change the execution of the operation by the setting the *lane* variable with the new value. Note, if the ego car is in the middle lane right and left lane changes are analyzed and the lane with the highest speed is taken.

```
421
422           //  change from left to middle lane
423
424           if (lane == 0) {
425            speed_lane = check_lane(sensor_fusion, car_s, lane, check_speed, 1,prev_size);
426            if (speed_lane > 0) {
427             lane = 1;
428            }
429
430            // change from right to middle lane
431           }
432           else if (lane == 2) {
433                speed_lane = check_lane(sensor_fusion, car_s, lane, check_speed, -1,prev_size);
434                if (speed_lane > 0) {
435                 lane = 1;
436                }
437              }
438
439              // change from middle to left or right lane
440
441              else {
442                   speed_lane_l = check_lane(sensor_fusion, car_s, lane, check_speed, -1,prev_size);
443                   speed_lane   = check_lane(sensor_fusion, car_s, lane, check_speed,  1,prev_size);
444                   if (speed_lane_l > speed_lane) {
445                    lane = 0;
446                   }

             else if (speed_lane_l < speed_lane) {
448                     lane = 2;
449                    }
450                    else if (speed_lane > 0) {
451                        lane = 0;
452                       }
453                 }
454
455           } // end change lane or emergency break
456
457          } // end too close
458
459         } // end in lane
460
```

- The **velocity adaption** module has been also added for the control of the velocity of the ego vehicle. It supports emergency breaking if the speed of the in-front is too small – see explanation above.

```
// ------------------------------------------------------------------------------------
//  velocity adaption
// ------------------------------------------------------------------------------------

// reduce velocity
if (too_close) {

  velocity -= VELOCITY_STEP;

  // emergncy break
  if (emergency_break) {
    velocity -= 2*VELOCITY_STEP;
  }
}
// increase velocity
else if (velocity < VELOCITY_MAX) velocity += VELOCITY_STEP;

```

- The **trajectory** architecture using the spline library (h file) has been reused from the Q&A with some code optimizations.

```
// ------------------------------------------------------------------------------------
// Trajectory
// ------------------------------------------------------------------------------------

// wavepoint list (x,y)
vector<double> ptsx;
vector<double> ptsy;

// start reference of vehicle
double ref_x    = car_x;
double ref_y    = car_y;
double ref_yaw  = deg2rad(car_yaw);

//  check if list size
if (prev_size < PREVIOUS_SIZE_LIMIT) {

  // take 2 points for path - kind of linarization (car_x + previous_x calculated from past linear)
  double prev_car_x = car_x-cos(ref_yaw);
  double prev_car_y = car_y-sin(ref_yaw);

  // build list

ptsx.push_back(prev_car_x);
  ptsx.push_back(car_x);
  ptsy.push_back(prev_car_y);
  ptsy.push_back(car_y);
}
else {

  // take reference from last entry of list
  ref_x = previous_path_x[prev_size-1];
  ref_y = previous_path_y[prev_size-1];
  // one more from past
  double ref_x_prev = previous_path_x[prev_size-2];
  double ref_y_prev = previous_path_y[prev_size-2];

  // calculate yaw
  ref_yaw = atan2(ref_y-ref_y_prev,ref_x-ref_x_prev);

  // build list
```

Cost & Parameters Tuning

To tune the model costs, the defines below have been specified.

```
// ------------------------------------------------------------------------
// New variables & constants
// + lane
// + velocity
// ------------------------------------------------------------------------

int    lane     = 1;    // start lane
double  velocity = 0;    // start velocity

// constants act as cost
const double VELOCITY_EMERGENCY  = 0.33; // speed of front car in terms percentance of ego velocity
const double VELOCITY_MAX       = 49.5;
const double VELOCITY_STEP      = 0.336;
const double VELOCITY_DEC       = 0.85;
const int    DISTANCE_NUM       = 50;
const double POINTS_PER_SEC     = 0.02;
const double REF_DISTANCE       = 30.0;
const double BACK_DISTANCE      = 0.33;
const int PREVIOUS_SIZE_LIMIT    = 2;
```

Further Optimizations

The following enhancements can optimize the model further:

1. Prediction of multiple lane change
2. Velocity control by PID, MCP controller
3. Optimization of collision check for lane change
4. Optimization of emergency breaking