The Model

The model used for this project is the kinematic model, which is a simplification of a dynamic model, since it ignores tire forces, gravity and mass.
The state of the model is defined by the following variables:

- x: position of the vehicle in forward direction.
- y: position of the vehicle in lateral (left) direction.
- psi: angle, orientation of the vehicle.
- v: velocity of the vehicle.
- CTE: cross-track-error; distance of the vehicle from the desired trajectory.
- EPSI: orientation error; difference between the angle of the vehicle orientation and the trajectory orientation.

We want to derive a model that captures how the state evolves over time and so we can provide inputs to change it. The inputs we use are given by the actuators.
The actuators include the following:

- delta: steering angle.
- a: acceleration (throttle pedal and brake pedal).

How the state changes over time is based on the previous state and the current actuator inputs. The following formulas were used to calculate the state update:

x_t+1 = x_t + v_t * cos(psi_t) * dt
y_t+1 = y_t + v_t * sin(psi_t) * dt
psi_t+1 = psi_t + (v_t / Lf) * delta * dt
v_t+1 = v_t +a_t * dt
CTE_t+1 = CTE_t + v_t * sin(EPSI_t) * dt [where CTE_t = f(x_t) - y_t)]
EPSI_t+1 = EPSI_t + (v_t / Lf) * delta_t * dt [where EPSI_t = PSI_t - desiredPSI_t]

(Lf is the distance between the front wheel of the vehicle and its center of gravity;
dt is the time step).

Polynomial Fitting and MPC Preprocessing

Before passing the data to the MPC model I transformed the waypoints (coming from the simulator) from the map coordinate system to the vehicle coordinate system.
This allowed to calculate the coefficients of a 3rd degree polynomial which was fitted to these vehicle points.
The initial state is therefore defined using these coefficients.
In particular, the value of x, y and psi are equal to zero (when not considering latency.
When considering latency then the x position takes into account the velocity multiplied by the latency).
The CTE at this initial state is calculated by evaluating the polynomial at x.
The orientation error (EPSI) is calculated from the first derivative of the polynomial at x.

Finally the state and the coefficients previously calculated are passed to the MPC solver, which returns the two new actuator values (steering angle and acceleration).

Cost function

The goal of the MPC is to provide values for the actuator that allow the vehicle to follow the desired trajectory, optimizing the solution.
The best trajectory is the one that minimizes the cost function.
The cost function includes several constraints, which help the optimizer to select the optimal solution.
In the cost function I have included:

- CTE.
- EPSI.
- Velocity (in order to avoid the vehicle to stop when the cost is equal to zero, penalizing it for not maintaining the reference velocity, which I set to 60 mph).

```
for (int i = 0; i < N; i++) {
  fg[0] += CppAD::pow(vars[cte_start + i] - ref_cte, 2);
  fg[0] += CppAD::pow(vars[epsi_start + i] - ref_epsi, 2);
  fg[0] += CppAD::pow(vars[v_start + i] - ref_v, 2);
}
```

- Control input delta, so to penalize the magnitude of the input.
- Control input a (acceleration).

```
for (int i = 0; i < N - 1; i++) {
  fg[0] += CppAD::pow(vars[delta_start + i], 2);
  fg[0] += CppAD::pow(vars[a_start + i], 2);
}
```

- Change-rate of the control inputs, in order to add temporal smoothness.
  In particular I added a multiplication factor, with an emphasis on delta.
  I started with a smaller value (around 50) but then increased it more, finally settling to 2000, which provided a very smooth car behavior.

```
for (int i = 0; i < N - 2; i++) {
  fg[0] += 2000 * CppAD::pow(vars[delta_start + i + 1] - vars[delta_start + i], 2);
  fg[0] += 5 * CppAD::pow(vars[a_start + i + 1] - vars[a_start + i], 2);
}
```

Timestep Length and Elapsed Duration (N & dt)

The MPC includes the prediction horizon, which is the duration over which future predictions are made.
The prediction horizon (T) is the product of N (number of time steps) and dt (the elapsed time between actuations).
The values of N and dt need to be fine tuned.
The general guideline is to have dt small and T as large as possible.
I started with N = 25 and dt = 0.05 which was not bad as the car drove very good in the initial part of the track, but then the model had problems in predicting the trajectory in the areas of tight curves.
I then reduced the value of dt to 0.03 which gave a bit of improvements, but the car was still behaving badly in sharp turns.
So I decided to decrease the val of N to 9, but the car was veering off quite a bit even in the initial part of the track.
I therefore increased the value of N, settling for 15.
Thus, the final values that I selected where N = 15 and dt = 0.03 and the car drove pretty good around the track with no erratic behaviors.

Model Predictive Control with Latency

The project has a built-in latency which is representing the delay present in a real-world car due to the fact that an actuation does not execute instantly, but there is a delay in the actuators dynamics.
Therefore if we don't take this into account the output values computed by the MPC solver (steering angle and acceleration) for the first time step are already late and refer to an action that should have been taken in the past (100 milliseconds earlier in our particular case). This causes the cross-track error to increase and the model to try to compensate for that, which ultimately causes the vehicle to swing, overshooting the desired trajectory.
To avoid this I embedded the latency factor into the initial car position, projecting the vehicle position forward by a factor of velocity*latency (so v* 0.1).