Code compilation:
Cmake works.

Model:
State: State contains 6 variables - position (x & y), psi (angle from x axis, +ve in anticlockwise direction), v (longitudinal velocity), cte (cross track error) and epsi (error in psi). Position and psi values are in vehicle's coordinate system, so +ve X axis is in car's longitudinal direction and +ve Y axis is towards vehicle's left.

Actuations: There are two actuations - delta (steering, +ve in anti-clockwise direction) and acceleration. Delta is in -25 to 25 range, normalized to -1 to 1 before sending the instruction to the vehicle. Acceleration is in -1 to 1 range (full deceleration to full acceleration).

Update equations:
x1 = (x0 + v0 * CppAD::cos(psi0) * dt);
y1 = (y0 + v0 * CppAD::sin(psi0) * dt);
psi1 = (psi0 + (v0/Lf) * delta0 * dt);
v1 = (v0 + a0 * dt);
cte1 = ((f0 - y0) + v0 * dt * CppAD::sin(epsi0));
epsi1 = ((psi0 - psides0) + (v0/Lf)*dt*delta0);

f0 = coeffs[0] + coeffs[1] * x0 + coeffs[2] * CppAD::pow(x0, 2);
psides0 = CppAD::atan(coeffs[1] + 2 * coeffs[2] * x0);

Timestamp length and elapsed duration:
N = 10
dt = 0.4

dt is choosen to be greater than 0.3 as 100ms latency and connection latency were coming around ~300ms. If we select a lower dt, then optimizer might select actuator values which are only valid for less than 300 ms and

ideally the car needs new actuator values before 300ms which is not possible with latency over 300ms

N is choosen to be 10 as we don't want to predict too far into the future, and optimization time also increases with increasing N.

I tried combinations where N ranged from 5 - 50 and dt ranged from 0.1 to 0.5. For dt values <= 0.3, car was not stable due to reasons described above. For 0.5, car was stable but it wasn't correcting the course enough around the bends (discretization error). 0.4 seemed best fit. Larger values of N didn't provide any significant gain and they don't make sense theoretically as well as it would not represent actual waypoints. For e.g the road might be turning but our way points are representing only straight section of the road, so larger N (and hence larger timestamp) is not useful.

Polynomial fitting and MPC pre-processing:
Preprocessing: velocity is converted to m/s (metres per second) and all waypoints are converted to car's coordinate system which makes it easier to display and makes MPC update equations easier. So the starting state of car always has its initial px, py, and psi as 0.

2nd degree polynomial is fitted to waypoints, after conversion to car's coordinate system.

Latency handling:
As described above, latency is handled by selecting appropriate value of dt.

Simulation:
Link to video: https://youtu.be/ltXvGzy1cCQ