The Model

I used model from the class.

$$x_{t+1} = x_t + v_t * cos(\psi_t) * dt$$

$$y_{t+1} = y_t + v_t * sin(\psi_t) * dt$$

$$\psi_{t+1} = \psi_t + \frac{v_t}{L_f} * \delta * dt$$

$$v_{t+1} = v_t + a_t * dt$$

Not sure what to comment on. Equations are implemented in file main.cpp from line 149.

Initially I tried to add velocity for X and ψ calculations, but that always lead to behaviour where car simply drove in circle. Circles are general issue with current implementation. If car increase speed from current 30mph it drives well, but when leaving the bridge controls go completely crazy. Bot sure how to debug, so input would be appreciated.

Non zero values that are passed to MPC

- **Velocity** = velocity + throttle value * latency
- **Cross tr err** = cte + velocity * sin(epsi) * latency
- **Epsi** = epsi + velocity * steering value / Lf * latency

Latency is set to target 0.1 seconds. Velocity is factoring in throttle, since that what it is about. Errors are just incorporating velocity into standard equations.

Timestep Length and Elapsed Duration (N & dt)

- **N = 12** empirically found sweet spot to match planned trajectory length. If I further increase this number predicted path will exceed planned path and that lead to 180 degrees turn.
- **dt = 0.1;** 0.2 does not work, as car moves too far and predictions got useless, bellow is pointless. Considering that we also have a latency of 0.1 overall fair number, however only for lower speeds.

Review Update

- Why smaller dt is better? (finer resolution)

Smaller time period will allow to put more dots to our planned path, so final result should be smoother. But there are more things co compute, so it is better to stop on some sweet spot.

- Why larger N isn't always better? (computational time)

Yes, time. Also this weird lag when you exceed the length of the planned path. In addition, predictions that are too far ahead are simply useless. We are not building the model that tracks historical values.

- How does time horizon (N*dt) affect the predicted path? This relates to the car speed too.

More measurements (N) with more time gap (dt) means further horizon, as we have more points with large space between.

Polynomial Fitting and MPC Preprocessing

I did not do any distinct preprocessing for MPC. Only transformed into the proper coordinate system with

```
for (int i = 0; i<ptsx.size(); ++i)      {
        double x = ptsx[i] - px;
        double y = ptsy[i] - py;
        rel_x[i] = x*cos(psi) + y*sin(psi);
        rel_y[i] = y*cos(psi) - x*sin(psi);
}
```

It is code from exercises, but I made some optimizations to cut down number of operations in the loop.

Model Predictive Control with Latency

Latency was implemented into the functions that were discussed above. In short, latency is used as a discount factor to all values that passed to MPC.

The vehicle must successfully drive a lap around the track.

2 laps.

https://youtu.be/kEoPQ3B9dLo

2 laps 60 mph after review

https://youtu.be/drFVQSxQUVI

Review comments

I incorporated suggestions, and they eventually worked, however I had to make some modifications to hyper parameters.

I decreased **N** to 10, since with a higher speed car plans too far during the turns and get out of the road.

I decreased weight in:

```
fg[0] += 1.0 * CppAD::pow(vars[a_start+i+1] - vars[a_start+i], 2);
```

From 1000 to 1. This changed nothing, but reviewer recommended so why not?

I increased weight in:

```
fg[0] += 10000.0 * CppAD::pow( vars[delta_start+i+1] - vars[delta_start+i], 2);
```

From 1000 to 10000, otherwise car had trouble to converge to the line and was wobbly. If you watch video, car in the beginning kind of shakes. With lower weight in happens longer and throughout the lap.

Also Px, Py and ψ are all getting updated.