# Track Maneuvering using MPC-Control


## Done by: Wael Farag


## 1. Objective

To design and implement a Model-Predictive-Control (PMC) controller for an autonomous vehicle to successfully maneuver around a complex track which has lots sharp turns. The controller receives measurements from the vehicle such as:

1. The “px” and “py” (the vehicle’s current x and y positions) measured in the global “map” coordinates.
2. The vehicle speed “speed” at the given instance measured in miles/hour (mph).
3. The vehicle orientation angle “*Ψ*” (-ve for left and +ve for right) in radians.
4. The vehicle orientation angle “*Ψ-unity*” in radians commonly used in navigation.
5. The current steering angle of the vehicle “steering_angle” measured in radians.
6. The current throttle value “throttle” given in the range [-1, 1] where (-ve for braking and +ve for speeding).
7. The `ptsx` and `ptsy` are two arrays that include the waypoints measured in global coordinates.

The MPC controller then uses some of the above information to produce a steer (angle) command to the vehicle in addition to a throttle command (speed).

## 2. The MPC Controller Implementation

The following are the steps used to implement the MPC controller:

1. The received waypoints (`ptsx` and `ptsy`) from the simulator are converted from global coordinates to vehicle coordinates using the transformation equations implemented in the code (“main.cpp” lines 118 => 130) using the received (also from the simulator) vehicle positions (“px” and “py”) and the vehicle orientation angle “Ψ”.

2. An $n^{th}$ order polynomial equation is then fitted using the transformed waypoints using the code in (“main.cpp” lines 132 => 144) (in our case n = 3). This polynomial now represents the “desired route” that the vehicle should follow precisely to find its way throughout the track.

3. The cross track error “cte” and the error in the steering angle “epsi” are the n calculated using the coefficients of the fitted polynomial as in (“main.cpp” lines 146 => 148).

4. The controller six states are then prepared and sent to the MPC solver function as in (“main.cpp” lines 159 => 163) to calculate the required “steering value” and “throttle value”.

5. The computed steering and throttle values are then sent to the simulator to command the car as in (“main.cpp” lines 182 => 185).

6. Within the MPC solver function, we first define the MPC controller horizon which is given by the time step (dt = 0.075 sec) and the duration in terms of the number of steps (N = 30) as in (“MPC.h” lines 14 => 16).

7. The number of controller states (N_STATES = 6) as well as the number of controller outputs (N_OUTPUT = 2) are also determined before as in (“MPC.cpp” lines 132 => 133).

8. The number of solver variables is then determined from the above information to be (n_vars = N_STATES*N + N_OUTPUT*(N-1)) as coded in (“MPC.cpp” lines 135 => 140).

9. Moreover, the number of solver constraints is then determined from the above information to be (n_constraints = N_STATES * N) as coded in (“MPC.cpp” lines 141 => 142).

10. The controller variables and constraints are the initialized as coded in (“MPC.cpp” lines 144 => 210).

11. The MPC objective function (to be minimized) is fully defined in the class coded in (“MPC.cpp” lines 21 => 119).

12. Then finally, the optimization solving algorithm (given by the IPOPT library) is then called (“MPC.cpp” lines 235 => 238) and the result is feed back to the simulator as mentioned before in step 5.

# 3. The MPC Objective Function

The objective function of the MPC controller is composed of several terms. Each term has its own sub-objective in the optimization problem. The main goal is to find a solution that can satisfy the purposes of these terms according to their weights (contribution) in the overall objective function. The final weights of all terms that determine their contribution are given in (“MPC.h” lines 22 => 31). The terms and their sub-objectives is listed as follows:

1. CTE term: The objective of the first term is to minimize the aggregation of cte’s for all prediction points (N=30) as coded in (“MPC.cpp” line 95).

2. eΨ term: The objective of this term is to minimize the aggregation of the errors “epsi” in the steering angle for all prediction points (N=30) as coded in (“MPC.cpp” line 96).

3. V error term: The objective of this term is to minimize the aggregation of the speed errors with respect with the reference speed (Vref = 100) for all prediction points (N=30) as coded in (“MPC.cpp” line 97).

4. Speed regulation term: The objective of this term which is coded in (“MPC.cpp” line 99) is to help the controller manage the speed throughout the track. The main purpose of this term is speed up

when the road is straight and to slow down when there is a turn ahead. The amount of slowing down is proportional to how sharp is the turn ahead.

5. Steer control term: The objective of this term which is coded in ("MPC.cpp" line 105) is to help the controller to optimize the control effort by not taking unnecessary sharp steering commands.

6. Acceleration control term: The objective of this term which is coded in ("MPC.cpp" line 106) is to help the controller to optimize the control effort by not taking unnecessary acceleration/braking commands.

7. Speed-steering term: The objective of this term which is coded in ("MPC.cpp" line 108) is to correlate between the steering command and the actual speed of the car. The idea is to let the controller when issues a relatively big steering command, to reduce the speed and vice versa.

8. Change of steering command term: The objective of this term which is coded in ("MPC.cpp" line 115) is to minimize the value gap between sequential steering actuations. In other words, reduce the sudden change in the subsequent steering commands.

9. Change of acceleration command term: The objective of this term which is coded in ("MPC.cpp" line 116) is to minimize the value gap between sequential acceleration actuations. In other words, reduce the sudden change in the subsequent acceleration commands.

Moreover, the constraints in the objective function are set as follows:

1. The initial points (point '0') of each controller state are initialized as the incoming state values from the simulator as coded in ("MPC.cpp" lines 37 => 43).

2. The rest of the points (from 1 => (N-1)) are constrained by the vehicle model which is coded in ("MPC.cpp" lines 74 => 80) and given by the following equations that are used to update the six states ($X_{k+1}$, $Y_{k+1}$, $\Psi_{k+1}$, $V_{k+1}$, $CTE_{k+1}$ & $e\Psi_{k+1}$):

$$X_{k+1} - (X_k + V_k \cos(\Psi_k)\, \Delta t) = 0$$
$$Y_{k+1} - (Y_k + V_k \sin(\Psi_k)\, \Delta t) = 0$$
$$\Psi_{k+1} - (\Psi_k - (V_k / L_f)\, \delta_k\, \Delta t) = 0$$
$$V_{k+1} - (V_k + acc_k\, \Delta t) = 0$$
$$CTE_{k+1} - ((f(x_0) - Y_k) + V_k \sin(e\Psi_k)\, \Delta t) = 0$$
$$e\Psi_{k+1} - ((\Psi_k - f'(x_0)) - (V_k / L_f)\, \delta\, \Delta t) = 0$$

where $f(x_0)$ and $f'(x_0)$ are the waypoints polynomial and the slope values at the point '0'.

# 4. The MPC Design Highligts

The following points need to be highlighted in the design process of the MPC:

1. Waypoints Polynomial: The waypoints polynomial coefficients are not used directly once they got calculated but instead they kind of pre-processed by taking the weighted averaged of their current

and previous values as coded in ("main.cpp" lines 164 => 177). This helps out to smooth transition between frames and makes the waypoints polynomial more stable.

2. Speed and the Objective function: The reference speed (Vref) is set at 100 mph which makes tuning the controller more challenging and the need for speed regulation around corners and at sharp turns is highly desirable. Therefore, two objective function terms are added to the overall MPC objective function: the "Speed regulation term" (#4) and the "Speed-steering term" (#7). The "Speed regulation term" specifically proved to be very effective as it make makes the speed inversely proportional the desired steer angle (which is large at sharp turns and make the car slower).

3. Actuator Latency: The actuators estimated latency of at least 100 msec. This has been compensated for by simply use future actuator commands instead of the first one calculated. For example, the MPC solver produces ((N-1)=29) predicted steering angle commands ($\delta_k \rightarrow \delta_{k+29}$). Instead of using $\delta_k$ as usual, I used $\delta_{k+3}$ as coded in ("main.cpp" lines 253 => 259) as 3*dt = 225 msec > 100 msec. I have tried $\delta_{k+1}$, $\delta_{k+2}$ and $\delta_{k+3}$ and the latter proved more effective.

4. Testing and Evaluation: extensive trials and errors endeavors are used to tune the many parameters of the MPC. However, to be more consistent and accurate, a numerical performance indicator need to constructed and coded as in ("main.cpp" lines 164 => 179). The Indicator is calculated by aggregating and averaging the $CTE^2$ and the $e\Psi^2$ over a period of 300 samples (enough to let the car drive for several laps around the track). The $e\Psi^2$ term is multiplied by 100 to have a comparable weight with the $CTE^2$. This method helped a lot to have more deterministic comparison between the different trials.

5. N and dt selection: On this point trials and errors are done on this part. I have tried dt from 0.05 $\rightarrow$ 0.2 and N from 5 $\rightarrow$ 30. I have reach a conclusion that a long sight for the controller is a good thing that improves its performance therefore, N = 30 is selected. In other words, if we need good predicted points in the near future (i.e. k = 1, 2 … 5), we have to let the controller solves for a long stride (i.e. N > 20). Moreover, dt needs to be small enough in order not to lose track on changes and big enough to allow for longer prediction strides. Finally, I selected 'dt' (=0.075 sec) to be smaller the actuator latency but big enough to have good prediction horizon.

# 5. Discussion and Conclusion

The following are some conclusive remarks on the project and the work done:

1. The MPC controller has a much more complex structure compared to the PID. However, it is more effective especially at higher speeds. It can handle issues like actuator latency and external disturbances much better.

2. The main problem with the MPC, is its tuning. There is no theory or criteria that proves that you have reached to the optimal value for the many hyper parameters used. All the methods of tuning are mainly based on extensive search with the incorporation of intuition and experience.

3. From my point of view, using transparent methods based on extensive “trial and error” endeavors guided by a numerical performance indicators is the best approach; as it lets you understand the problem at hand much deeper. Furthermore, it allows you to incorporate your intuition and experience which reduces a lot of the search space; and consequently allows you at the end to be more effective.

4. The most powerful aspect in the design of the MPC is the tailoring of the objective function. It gives a great flexibility to the designer to balance between conflicting requirements and make a trade-off.

# 6. Suggested Improvements

The following list summaries the suggested improvements:

1. Tring another term to may be added to the MPC objective function which is “maintain δ*acc = constant”. The idea at big steering angle the car is going to a sharp turn and needs to slow down and vice versa.

2. I believe one set of hyper-parameters will not be enough for all the range of speed. Therefore, we may need a kind of adaptive MPC where it has several sets of parameters for each speed range (similar to gain scheduling in PID).

3. Need to add or invent several other performance indicators, like a one to track the time in which the car is able to a complete one lap. Another suggested one is to track overshoots and under shoots from the road center … etc.