Linearized MPC for Lateral Control

Mustafa Poonawala
January 7, 2022

1 SUMMARY

In this project we implement a Lateral MPC controller for path tracking. In MPC, at each sampling time step, starting at the current state, an open-loop optimal control problem is solved over a finite prediction horizon. The optimal control input sequence for the control horizon is found by the numerical solver. The optimal control input sequence minimizes the objective function. The first element of the optimal control input sequence is applied to the system (vehicle). At the next time step, a new optimal control problem is solved over a shifted horizon based on the then 'current state' of the system and the new optimal control input sequence is found. This process continues till the goal is reached.
As the first iteration or a baseline, we implement the controller using a linearized kinematic bicycle model and formulate the optimization problem as a Quadratic Program. In this iteration, the longitudinal aspect is not included and the MPC is implemented for a constant vehicle speed. The controller is implemented in C++ in a ROS node. CasADi is used to formulate the problem and IPOPT solver is used to solve the problem.

2 MPC FORMULATION

The MPC equations are as follows.

1. System model:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v \times \cos\psi \\ v \times \sin\psi \\ \frac{v}{L} \times \tan\delta \end{bmatrix} \tag{2.1}$$

   Where x,y and ψ represent the pose of the vehicle in the global frame and are the states of our system model. v represents the velocity of the vehicle and is considered a constant.

We keep it 1m/sec. δ is the steering angle and represents the input to our system. Thus, the goal of our MPC solver is to find the optimal sequence of δ's to track the path.
It can be seen that the system model is non linear and thus cannot be expressed in the standard "$\dot{X} = AX + BU$" form. To formulate our problem as a Quadratic Program we need a linear model of our system. We thus need to linearize the non linear model about a suitable point.
We choose the current state as the point to linearize around and use Euler's approximation to get the linear model as follows. We make the assumption that our linear model is valid over the prediction horizon and thus we can use the linear model in our MPC formulation.
First, discretizing the continuous model as follows:

$$\begin{bmatrix} x[k+1] \\ y[k+1] \\ \psi[k+1] \end{bmatrix} = \begin{bmatrix} x[k] \\ y[k] \\ \psi[k] \end{bmatrix} + \begin{bmatrix} v \times \cos\psi[k] \\ v \times \sin\psi[k] \\ \frac{v}{L} \times \tan\delta[k] \end{bmatrix} \times Ts \tag{2.2}$$

where Ts is the sampling time.
Now, we linearize around the 'current state' denoted by subscript 'o' as follows:

$$\begin{bmatrix} x[k+1] \\ y[k+1] \\ \psi[k+1] \end{bmatrix} = \begin{bmatrix} x[k] \\ y[k] \\ \psi[k] \end{bmatrix} + \begin{bmatrix} v \times \cos\psi_o - v \times \sin\psi_o \times \Delta\psi \\ v \times \sin\psi_o + v \times \cos\psi_o \times \Delta\psi \\ \frac{v}{L} \times \tan\delta_o + \frac{v}{L} \times (\sec\delta_o)^2 \times \Delta\delta \end{bmatrix} \times Ts \tag{2.3}$$

It can be seen that for a constant v, ψ_o, δ_o, Ts the system is linear, thus we have created a linearized bicycle model of the vehicle which we can use to formulate a quadratic program.

2. Reference path:
For a reference path, we rely on a separate path planning algorithm which could be anything from Dijkstra to RRT* to generate our reference path as a sequence of waypoints. The waypoints are defined as x and y co-ordinates in the global frame.

3. Objective Funtion:
We define the objective function as follows:

$$J = \min_{\delta[i]} \left(\sum_{i=1}^{N} e_x^2[i] + e_y^2[i] + w \times \delta^2[i] \right) \tag{2.4}$$

$$e_x[i] = x[i] - x_{ref}[i] \tag{2.5}$$

$$e_y[i] = y[i] - y_{ref}[i] \tag{2.6}$$

where $x_{ref}[i], y_{ref}[i]$ are the co-ordinates of the waypoints that the car needs to track at instant [i] and w is a weight which allows us to tune the aggressiveness of the controller by placing more or less weight on the input. Thus, the objective is quadratic.

4. Constraints:
The following are the constraints that we include:

a) The system dynamics need to be included as follows.

$$\begin{bmatrix} x[i+1] \\ y[i+1] \\ \psi[i+1] \end{bmatrix} = \begin{bmatrix} x[i] \\ y[i] \\ \psi[i] \end{bmatrix} + \begin{bmatrix} v \times \cos\psi_o[k] - v \times \sin\psi_o[k] \times \Delta\psi \\ v \times \sin\psi_o[k] + v \times \cos\psi_o[k] \times \Delta\psi \\ \frac{v}{L} \times \tan\delta_o[k] + \frac{v}{L} \times (\sec\delta_o[k])^2 \times \Delta\delta \end{bmatrix} \times Ts \tag{2.7}$$

Where N_p is the prediction horizon and $i = 1...N_p$ creates N_p constraints for the dynamics over the prediction horizon and $\psi_o[k], \delta_o[k]$ are the points about which we have linearized our model at the time instant k.

b)

$$-\delta_{max} <= \delta[i] <= \delta_{max} \tag{2.8}$$

Equation (2.8) represents the constraints on the input over the entire prediction horizon where δ_{max} is the absolute value of steering angle possible in either directions.[1]

3 Implementation

The MPC controller has been implemented as a ROS node. The commented code and the launch file can be referred for detailed understanding. Link to repository. The MPC controller node consists of the following publishers and subscribers:

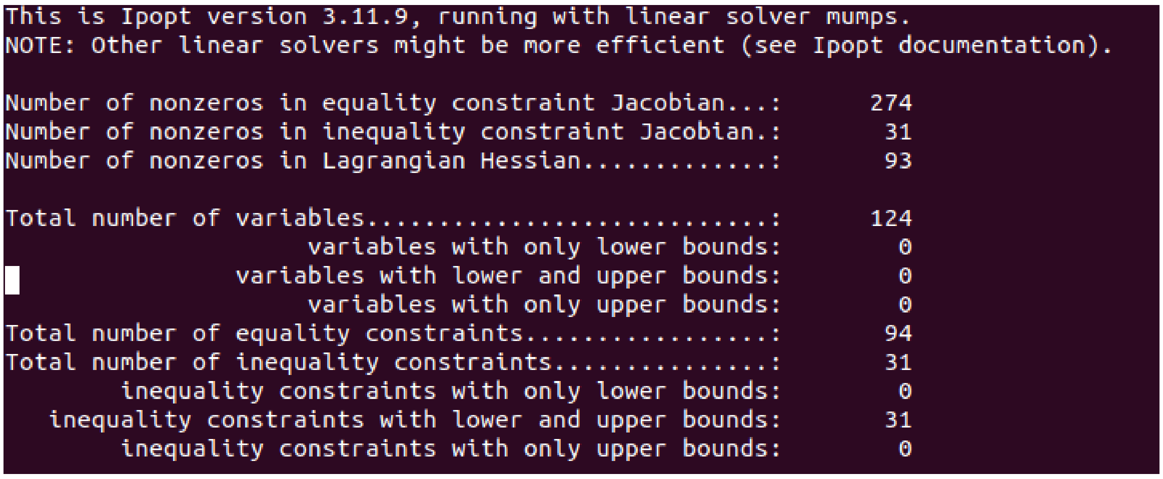
1. Car pose subscriber: This subscriber listens to the appropriate topic to get the updated pose of the vehicle from some localization algorithm. We use a particle filter node.
2. Path subscriber: The subscriber listens to the appropriate topic to collect the reference path generated by any path planning algorithm.
3. Drive command publisher: This publishes the optimal steering angle command to the appropriate topic for the actuator to pick up from.

The optimization problem is framed using CasADi and solved using IPOPT. Both CasADi and IPOPT are open source and thus chosen for the project. CasADi installation guide and user guide pdf can be found online. We have used CasADi's Opti stack which is an abstraction to formulate the problem, since it makes the formulation much more human readable and less tedious with very little performance overhead.

4 Results:

1. We implemented the controller and tested the performance in the F1Tenth simulator. The solver takes around 23 ms on average to solve the optimization problem on a home laptop. This would allow us the sampling frequency of 43Hz. As we would not get such good performance on the Nvidia Jetson platform, to get a more realistic idea, we purposely reduce the sampling frequency to 5Hz using the RATE class of ROS and see appropriate path tracking at a speed of 1m/sec.

[1] We can also include rate constraints on the steering command δ and performance can be explored by including them

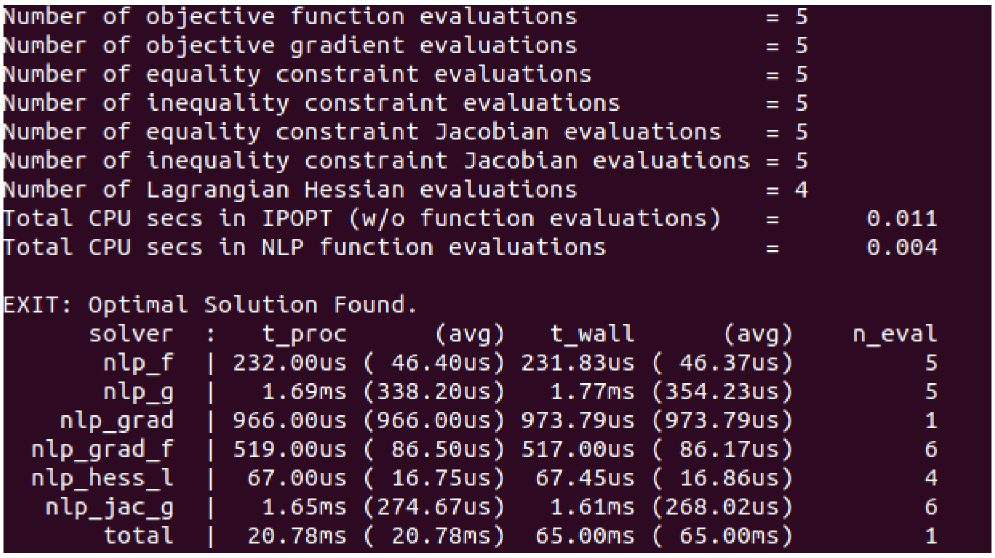


```
This is Ipopt version 3.11.9, running with linear solver mumps.
NOTE: Other linear solvers might be more efficient (see Ipopt documentation).

Number of nonzeros in equality constraint Jacobian...:      274
Number of nonzeros in inequality constraint Jacobian.:       31
Number of nonzeros in Lagrangian Hessian.............:       93

Total number of variables............................:      124
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:       94
Total number of inequality constraints...............:       31
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:       31
        inequality constraints with only upper bounds:        0
```

Figure 4.1: Screen grab of the solver output for an MPC iteration.



```
Number of objective function evaluations             = 5
Number of objective gradient evaluations             = 5
Number of equality constraint evaluations            = 5
Number of inequality constraint evaluations          = 5
Number of equality constraint Jacobian evaluations   = 5
Number of inequality constraint Jacobian evaluations = 5
Number of Lagrangian Hessian evaluations             = 4
Total CPU secs in IPOPT (w/o function evaluations)   =      0.011
Total CPU secs in NLP function evaluations           =      0.004

EXIT: Optimal Solution Found.
      solver  :   t_proc      (avg)   t_wall      (avg)    n_eval
       nlp_f  | 232.00us ( 46.40us) 231.83us ( 46.37us)         5
       nlp_g  |   1.69ms (338.20us)   1.77ms (354.23us)         5
    nlp_grad  | 966.00us (966.00us) 973.79us (973.79us)         1
  nlp_grad_f  | 519.00us ( 86.50us) 517.00us ( 86.17us)         6
  nlp_hess_l  |  67.00us ( 16.75us)  67.45us ( 16.86us)         4
   nlp_jac_g  |   1.65ms (274.67us)   1.61ms (268.02us)         6
       total  |  20.78ms ( 20.78ms)  65.00ms ( 65.00ms)         1
```

Figure 4.2: Screen grab of the solver output for an MPC iteration.

5 FUTURE SCOPE:

The following could be pursued next:

1. We can implement the controller on the actual vehicle and come up with more formal performance metrics.

2. Different solvers can be used and performance can be compared. Example IPOPT V/s qpoasis.

3. The tuning parameters can be varied and the effects on performance can be analyzed

which can help in fine tuning the controller.

4. The effect of removing the input constraints from the optimization formulation can be investigated.
5. Velocity can be made a decision variable instead of keeping it fixed and a non linear program can be investigated.
6. Obstacle avoidance aspect can be added in the MPC formulation using feedback from the LIDAR.

References:

1. CasADi installation instruction.
2. CasADi user guide
3. CasADi Opti stack tutorial.
4. Linear MPC reference report.
5. Berkley Lab MPC book.
6. Project repository.