REAL-TIME ANALYSIS FOR NONLINEAR MODEL PREDICTIVE CONTROL OF AUTONOMOUS VEHICLES

Muhammad Awais Abbas, J. Mikael Eklund and Ruth Milman

Faculty of Engineering and Applied Science,
University of Ontario Institute of Technology, Oshawa, ON L1H 7K4

ABSTRACT

This paper presents an online Nonlinear Model Predictive Control (NMPC) framework for trajectory tracking of autonomous vehicles. The operating environment is assumed to be unknown with various different types of obstacles. A bicycle model is used for the prediction of the future states in the NMPC framework, and a fully nonlinear CarSim vehicle model is used for the simulations. Real-time analysis is presented for a particular situation and the effect of warm initialization of optimization process on the computation time is elaborated. Simulation results show that the NMPC controller provides satisfactory online tracking performance while satisfying the real-time constraints, and warm initialization reduces the optimizer computational load significantly.

Index Terms— NMPC, CarSim, Autonomous, Control, Trajectory, Real-Time.

I. INTRODUCTION

The worldwide interest on autonomous vehicles (airborne, space-borne, ground and submersible) has been increasing rapidly. Autonomous ground vehicles have found wide range of applications ranging form mine clearance to saving human lives in hostile environments. Model predictive control is one of the most common method used for the trajectory tracking of an autonomous vehicle.

Many unmanned vehicles in use today are not autonomous, but rather remote piloted vehicles. Numerous efforts have been made to enable the truly autonomous operation of vehicles. Sutton [1] considered the problem involving a constrained submarine in order to explore the practicalities of the NMPC problem implementation. Online gradient optimizer followed by an extended Kalman Filter state estimator was used to control a continuous time submarine system while the controller embedded a discretized nonlinear model. Kim [2] investigated the feasibility of nonlinear model predictive tracking control (NMPTC) for autonomous helicopters. NMPTC algorithm was formulated for planning the paths under input and state constraints, and was implemented online using gradient-descent method. Eklund [3] proposed a supervisory controller for pursuit and evasion of two fixed-wing autonomous aircrafts. NMPTC was used for real-time trajectory tracking of evader as well as pursuer aircraft. The NMPTC controller was then integrated in a UAV which participated in Pursuit Evasion Games against a US Air Force Pilot operated F-15 aircraft. Fahimi [4] designed NMPC law for controlling multiple autonomous surface vessels in arbitrary formations within environments containing obstacles. Vougioukas [5] used NMPTC for precision guidance of agricultural tractors to assist farmers in their day to day activities. Presently, alot of researchers are focusing on active steering methods to reduce the number of motor vehicle accidents and make the driving experience safer. This paper presents such a predictive controller which can steer away a vehicle from unknown obstacles in real-time.

This paper is organized as follows: Section II presents the mathematical formulation of the MPC framework. Section III presents the results obtained from the simulation of a ground vehicle in a realistic environment, then some light is shed on the real-time aspects of the controller implementation. Finally, Section IV concludes the paper after discussing the results form the presented simulation scenario.

II. TRAJECTORY TRACKING USING MPC

The trajectory tracking architecture (see Figure 1) used for the control of ground vehicles has long been used in the aerospace systems where it is known as Guidance and Navigation Control (GNC) system [6].

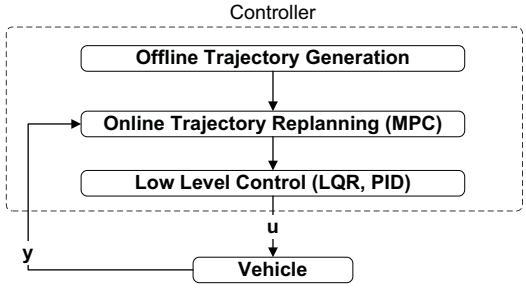

Fig. 1: Vehicle Guidance System Architecture

In Figure 1, the top layer is the offline trajectory generation layer. The trajectory is generated in an offline manner only once before the start of the mission. At runtime, the

2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)
978-1-4673-1433-6/12/$31.00 ©2012 IEEE

trajectory needs to be replanned on the occurrence certain events such as: an obstacle is detected on the pre-calculated trajectory. This replanning happens on the second layer where the most intelligent and computationally demanding controller (MPC) is present to regenerate the optimized trajectory and steer the vehicle efficiently through the driving space. The third layer is the low level control layer which contains the basic control algorithms like PID and LQR to perform low level control on actuators (brake, throttle etc) based on the command from the higher level control.

II-A. Vehicle Modeling

The main requirement in any model predictive control (MPC) application is the plant model. MPC optimizer uses this model to predict the future plant behavior and to plan an optimized trajectory. At each control interval an MPC algorithm attempts to optimize the plant behavior by computing a sequence of future manipulated variable adjustments. Multiple iterations of the algorithm are required, so the mathematical model needs to be solved repeatedly, thus it forms a large portion of the computation load. High fidelity math models of a four wheeled-vehicle can have over 110 differential equations [7]. Such a model can be very hard to implement in real-time due to the amount of the computation power required to predict the states.

However, the mathematical model can be significantly simplified by making some basic assumptions like non-slippage of tires, planar motion and linear tire characteristics and considering the model as a *bicycle model* [8]. In a bicycle model, the front left and front right wheels are lumped into one wheel. Therefore the steering angle of the front left and front right wheel can be resented by a single angle, δ_f. The five states of the vehicle are defined as: (X, Y) are the position coordinates of the vehicle in an inertial frame, ψ is the vehicle yaw angle or heading, $\dot{\psi}$ is the yaw rate and β is the vehicle's side-slip angle. The dynamic model can be written as a nonlinear function of the state vector $\xi(t)$ and the input vector $u(t)$:

$$\begin{aligned} \dot{\xi}(\mathbf{t}) &= \mathbf{f_{cont}}(\xi(\mathbf{t}), \mathbf{u}(\mathbf{t})) \\ \eta &= \mathbf{C}.\xi \end{aligned} \tag{1}$$

where $u(t) = \delta_f$, $\xi = [\beta\ \psi\ \dot{\psi}\ X\ Y]'$ and C is a filter matrix which eliminates the states not required for path tracking. In this case, the output η is set to track the X and Y coordinates of the vehicle in the inertial frame. C matrix becomes:

$$C = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

with this choice of C matrix the output states become

$$\eta = [X \quad Y]^T$$

II-B. Cost Function

To generate the trajectory with our required specifications, we define the following generic cost function:

$$J = \phi(\tilde{\eta}_N) + \sum_{k=0}^{N-1} L(\xi_k, \tilde{\eta}_k, u_k) \tag{2}$$

where $\phi(\eta_N) = \frac{1}{2}\eta_N^T Q_0 \eta_N$ is the terminal cost function and penalizes the trajectory deviation at the last time step ($k = N$) of the look-ahead horizon. $\tilde{\eta}_k = \eta_{d,k} - \eta_k$ is trajectory deviation variable where $\eta_{d,k}$ is the desired trajectory and η_k is the vehicle's anticipated trajectory. $L(\xi_k, \eta_k, u_k)$ is the running cost and is defined as:

$$L(\xi_k, \tilde{\eta}_k, u_k) = \frac{1}{2}(\tilde{\eta}_k^T Q \tilde{\eta}_k + \xi_k^T S \xi_k + u_k^T R u_k) \tag{3}$$

The running cost penalizes the trajectory deviation at each time step ($k = (1, ..., N-1)$) during the entire horizon. The cost function (Equation 2) is augmented to suit our needs. The matrices Q_0, Q, R and S are the penalty weighing matrices. The matrices $Q_0(2 \times 2)$ and $Q(2 \times 2)$ penalize the terminal trajectory deviation and the running trajectory deviation respectively. The matrices $S(5 \times 5)$ and $R(1 \times 1)$ penalize the large state values and the large input values. All the penalty weighing matrices are diagonal matrices.

For obstacle avoidance an easily differentiable, point-wise repulsive potential function P_k is used:

$$P_k = \frac{1}{(x - x_0)^2 + (y - y_0)^2 + \epsilon} \tag{4}$$

where (x, y) is the current location of the vehicle and (x_0, y_0) is the location of the obstacle to be avoided. ϵ is a small positive number for non singularity. Fahimi [4] and Xi [9] used such potential function for obstacle avoidance. A constraint term is introduced to account for the actuator saturation. With this choice of potential function, the running cost becomes:

$$L(\xi_k, \tilde{\eta}_k, u_k) = \frac{1}{2}(\tilde{\eta}_k^T Q \tilde{\eta}_k + \xi_k^T S \xi_k + u_k^T R u_k) + P_k + constraint \tag{5}$$

The cost function is minimized by using the gradient descent optimization method [1] which minimizes the cost function by taking successive steps towards the downards slope of the cost function at each optimizer iteration.

III. SIMULATION RESULTS

III-A. Simulation Environment

MATLAB was used for the NMPC implementation because CarSim math mode can be extended to Simulink/MATLAB in-order to add external driver models. CarSim vehicle model was exported as an S-function into the Simulink model. NMPC controller was coded as MATLAB script file and imported to the Simulink by using the MATLAB Function block (Figure 2). Sampling time of the NMPC controller was set

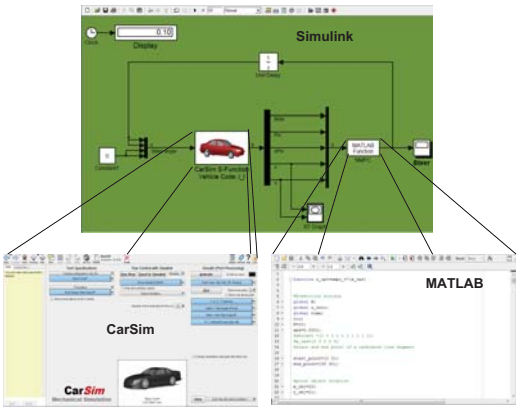

Fig. 2: Simulation environment

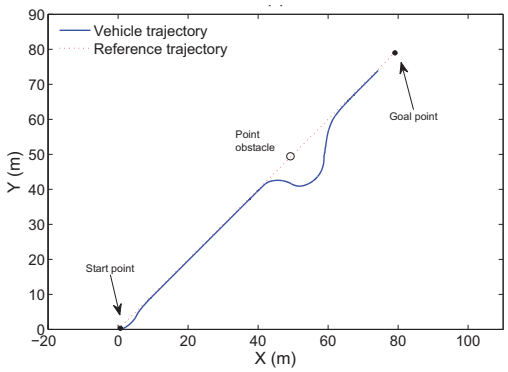

Fig. 3: Vehicle trajectory with a point obstacle

by changing the sampling time value in the Function Block parameter. All simulations were performed using the fully nonlinear CarSim vehicle model. An IBM ThinkPad Laptop Computer with Intel® Core™i7-2620M 2.27GHz dual core processor, 4GB RAM and running 32-bit Windows® 7 Operating System was used for the simulations.

III-B. Obstacle Avoidance Simulation

A tests was conducted to demonstrate the NMPC controller's ability to steer the vehicle in an *unknown environment* with obstacle. The vehicle started from South West corner, $(X, Y) = (0, 0)$, and the goal point was located on the North East corner $(X, Y) = (80, 80)$, of the simulation area (see Figure 3). Reference trajectory was a straight line joining the start point to the goal point. A point obstacle was placed on the planned path of the vehicle and the controller parameters were: Horizon Length $N = 20$ *steps*, Sampling Time $T_s = 0.05s$, Steering Angle Limit = $-20\,\text{deg} \leq u \leq 20\,\text{deg}$, Longitudinal Speed $v_x = 5.5m/s$, Simulation Time $20s$, $Q_0 = Q = [0.1\ 0; 0\ 0.1]$, R=[0.01], S=$0(5 \times 5)$. It was seen that the vehicle clearly avoided the obstacle by steering away from the obstacle and reaching the goal point successfully.

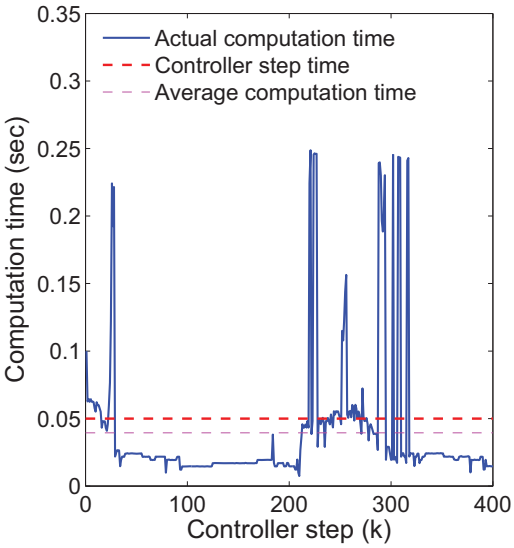

Fig. 4: Cold start: Computation time for each controller time step

III-C. Real-time Analysis

For real-time implementation it is very important that the whole controller processing is completed within one controller time step. Delayed solution can destabilize the system so it is preferable to have a suboptimal solution rather that an optimal and delayed solution. Time per controller step (k) is an important performance metric which tells us whether if the NMPC algorithm is suitable for online and real-time application. It is evident from the previous simulation results that the computation time is variable and depends upon the simulation scenario. If there is an obstacle in range or the vehicle is away from the desired trajectory, the computation time per controller step increases. Also, it is worth mentioning here that these time measurements are highly variable depending on the processor speed and are presented here for reference purposes only.

During the simulations, the computation time per controller step and the number of iterations per controller step were stored and plotted. Figure 4 shows the computation load for a typical simulation scenario by initializing the input vector u^* with zero at each controller step (*cold start* method). It was seen that the computation time exceeded the controller step size $(0.05s)$ multiple times and the average computation time was closer to the time limit boundary.

When the same test was performed with the identical controller parameters and tuning matrices but initializing the input vector with previous step's input vector (*warm start* method), the average computation time decreased visibly as compared to previous test (see Figure 5).

IV. DISCUSSION ON RESULTS

The same test was performed 10 times and average results were compared in Table I. Its is evident that initializing

Table I: Comparison between cold start and warm start methods

Parameter	Cold Start	Warm Start	Comparison
Total optimizer iterations for a $20s$ run	6539	4410	32.5% decrease in iteration count
Avg. computation time per controller step	$0.0393s$	$0.0275s$	30% decrease in avg. computation time per controller step
Number of times $0.05s$ limit is violated	70	38	45% decrease in controller limit violations

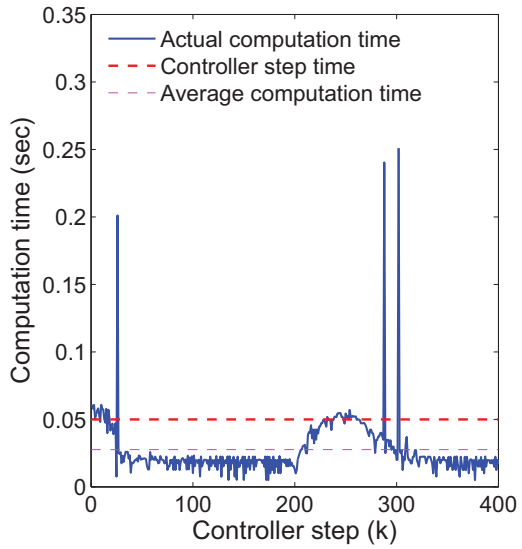

Fig. 5: Warm start: Computation time for each controller time step

optimization process with previous optimization results decreased iteration count significantly (32.5%) thus aiding the gradient algorithm for real-time application. Still there were occurrences of the controller limit violations in warm start case. Large computation time steps occurred at complicated points of the simulation such as when an object is encountered on the path or the vehicle is steering to approach the reference trajectory. These violations can be avoided by limiting the iteration count to a certain value so the optimization loop terminates as soon as that iteration number is reached. In these simulations, the time for each algorithm iteration can be found by dividing the time taken for computation of each controller step by the number of iterations per controller step ($time\ \ per\ \ iteration = \frac{computation\ \ time}{iteration\ \ count}$).

In this particular case the time per iteration count was found to be $0.0025s$. Which means the optimizer could iterate for a maximum of $\frac{0.05}{0.0025} = 25\ times$ per controller time step without exceeding the $0.05s$ time ceiling. Thus if a strict real-time implementation is intended, a counter can be used which terminates the loop if iteration count increases 25 thus keeping the overall processing time within $0.05s$. However, limiting the iteration count can cause stability problems, the effect of which is not considered in this research and is left for future work.

V. REFERENCES

[1] Gordon J. Sutton and Robert R. Bitmead, "Performance and computational implementation of nonlinear model predictive control on a submarine," in *Nonlinear Model Predictive Control*, Frank Allgwer, Alex Zheng, and Christopher I. Byrnes, Eds., vol. 26 of *Progress in Systems and Control Theory*, pp. 461–472. Birkhuser Basel, 2000.

[2] H.J. Kim, D.H. Shim, and S. Sastry, "Nonlinear model predictive tracking control for rotorcraft-based unmanned aerial vehicles," in *American Control Conference, 2002. Proceedings of the 2002*, 2002, vol. 5, pp. 3576 – 3581 vol.5.

[3] J. M. Eklund, J. Sprinkle, and S. S. Sastry, "Switched and symmetric pursuit/evasion games using online model predictive control with application to autonomous aircraft," *Control Systems Technology, IEEE Transactions on*, vol. PP, no. 99, pp. 1 –17, 2011.

[4] F. Fahimi, "Non-linear model predictive formation control for groups of autonomous surface vehicles," *International Journal of Control*, , no. 8, pp. 1248–1259, Aug. 2007.

[5] Y. Vougioukas, S. Ampatzidis, "A nonlinear model predictive path tracking for precision guidance," *In: Proceedings of the XVI CIGR World Congress (International Commission of Agricultural Engineering), Germany*, vol. 1958, 2006.

[6] P. Falcone, F. Borrelli, H.E. Tseng, J. Asgari, and D. Hrovat, "A hierarchical model predictive control framework for autonomous ground vehicles," in *American Control Conference, 2008*, june 2008, pp. 3719 – 3724.

[7] Mechanical Simulations: CarSim Documentation, "CarSim Manual: Math Models," *Accessed: January 18, 2012*.

[8] Danwei Wang and Feng Qi, "Trajectory planning for a four-wheel-steering vehicle," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, 2001, vol. 4, pp. 3320 – 3325 vol.4.

[9] Wei Xi and John S. Baras, "Mpc based motion control of car-like vehicle swarms," *Mediterranean Conference on Control and Automation, Athens, Greece*, 2007.