Model Predictive Control for Automated Vehicle Steering

Ahmad Reda, Ahmed Bouzid, József Vásárhelyi

University of Miskolc, Institute of Automation and Info-communication, Egyetemváros, 3515 Miskolc, Hungary
{autareda, qgebouzid, vajo}@uni-miskolc.hu

Abstract: The autonomous vehicle steering system, a multi-input multi-output (MIMO) system, is challenging to design using traditional controllers due to the interaction between inputs and outputs. If PID controllers are used the control loops are executed independently of each other as there is no interaction between the loops. Designing a larger system increases the controller parameters requiring tuning. Model Predictive Control (MPC) overcomes this problem, as it is a multi-variable control method taking into account the interactions of the variables in the target system. Achieving a high safety level is also critical for autonomous vehicle systems. This can be provided by an MPC controller, which can handle constraints such as maintaining a safe distance from other cars. Wider applicability of the Model Predictive Controller calls for more efficient hardware architectures for implementation. The aim of this paper is to achieve optimal implementation of the MPC controller by increasing the computational speed in order to reduce execution time for optimization. An MPC controller is used to control the steering system of an autonomous vehicle to keep it on the desired path. A traditional MPC controller is used to control the system where the plant dynamics do not change, whereas an Adaptive MPC controller is used when the system is nonlinear or its characteristics vary with time (the longitudinal velocity changes as the vehicle moves). Results are discussed in terms of performance, resource utilization, cost, and energy-effective implementations taking into consideration a reasonable size number of constraints handled by the controller.

Keywords: Autonomous Vehicle; Steering System; Model Predictive Control (MPC); Field Programmable Gate Array (FPGA); System on Chip (SOC)

1 Introduction

In recent years, research in the automotive industry has been growing in order to address the challenges of this application domain. Automotive control applications require high performance and cost reduction at the same time [5]. The control system requirements are becoming higher, and to achieve the improvement in control performance, the optimization process is incorporated into the control system design. The optimization process is subject to an increased number of factors, such as physical, safety, and economic constraints (power consumption, actuator saturation, etc.). In this context, Model Predictive Control (MPC) is a powerful optimization strategy for feedback control based on the model of the system. Basically an MPC controller runs a set of forecasts forward in time on the system model for different actuation strategies. MPC determines the immediate next control action based on the optimization. Next, it reinitializes the optimization in order to define the next control input [7]. The current and future control inputs are determined based on minimizing the difference between the target setpoint and the predicted output [13]. MPC features and capabilities are very effective in terms of meeting the requirements and achieving the optimization tasks. A basic MPC controller solves Linear Programming (LP) problems, which can be formulated as quadratic programming (QP) problem [12]. Also, the MPC controller has a natural capability to handle soft and hard constraints. That means, the requirements that are imposed by the operating conditions can be managed and formulated using the constraints. However, MPC controller implementation has several challenges such as high computational load and high power consumption, whereas the embedded system applications have limitations in their hardware resources.

One of the most effective solutions in order to achieve MPC implementations for embedded system applications which have constraints related to the computational time, is the use of the hardware acceleration. In this context, the deployments of an embedded MPC controller can achieve using reconfigurable hardware such as Field Programmable Gate Array (FPGA) or System on Chip (SoC), which is popular due to its high computational capabilities, parallel processing and development framework [11]. In this context, the main contributions of this paper are the study and the analysis of the efficiency of implementing control methods, in addition to the use of rapid prototyping methods (here hardware/software co-design using Embedded Coder and HDL Coder) for the implementation of embedded systems dedicated for digital signal processing considering performance, execution time and resources consumption. The research applied functional on-target rapid prototyping using Embedded Coder and HDL coder. The suggested implementation method is based on taking the optimization problem of the control method through MATLAB Simulink, Fixed-Point Designer, Embedded Coder and HDL coder. The suggested method allows the authors to focus on the verification, the validation and the test of the embedded system rather than programming, which in turn gives the ability to refine the design, tune the MPC controller parameters and see the results in the real-time. Finally, different optimization strategies were implemented and the obtained results were compared in terms of reducing the execution time and hardware resources consumption.

FPGA based systems have been applied for a variety of applications, such as image and signal processing, aerospace, energy, autonomous vehicles, telecommunications (5G) and medical field. In paper [1] an analytical study for Adaptive MPC controller under external disturbances signals was provided, the Lipschitz-based approach was used and provides satisfactory stability and robustness. Saragih et al. used the MPC controller for visual-based control system application (face tracking system) to control the motion of a robot, where the MPC controller was implemented to control the camera movements in order to keep the tracked face at the center of the camera – see [21]. Paper [4] provides an overview of a real-time optimization problem for automotive and aerospace applications with a focus on MPC controller. The optimal control problem was formulated based on the cost function and the system constraints, in addition, numerical algorithms and their implementations on an embedded computing platform were discussed. The improvement of fuel economy for power-split hybrid electric vehicles (HEV) was discussed in [2]. The energy management system was formulated as a nonlinear and constrained system. The MPC controller was used to split the power between the combustion engine and electrical machines at the different system operating conditions. The proposed approach provided an improvement compared to the controllers in commercial Powertrain System Toolkit (PSAT) software. The research reported in [8], proposed a control approach based on combining steering and braking MPC controllers. The authors in the paper introduced two model predictive controllers. The first one was implemented on a four-wheel vehicle model which determines the steering angle and braking torques to track the desired trajectory. The second MPC controller was implemented on a simplified bicycle model with a smaller number of inputs. The obtained results showed that the first controller provides good performance in terms of tracking the reference trajectory at low and high-speed, but the computation was time-consuming. On the other hand, the second controller showed unsatisfactory performance at high speed due to the simplicity of the vehicle model [8].

Paper [24] presented research of edge cloud on the Internet of Things (IoT) where the Model Predictive Controller evaluates the system properties. The paper presented the potential of merging the IoT, 5G, and cloud computing with the efficiency of deploying the automatic control system for time-sensitive and mission-critical processes. Haidegger et al. in [20] stated that the predictive and model-based control gives satisfactory performances only in the case of providing the accurate system's behavior and cascaded control approach. An empirical design with the use of Smith predictor for a telesurgical robot system was suggested in order to deal with the large latencies. In the same context of paper [20], the article [10] suggested a cascaded control structure to deal with the time delay in a teleoperation robot system. The suggested method used the extended Kessler's method sported by a predictive control method. Fuzzy–PID controller was also suggested to improve the performance. Using the extended Kessler's method with Smith predictor provides good control. MPC controller deals with linear-time-invariant (LTI) plant model, which allows predicting the future behavior of the system [22]. Nevertheless, paper [17] suggested a strategy to control heterogeneous traffic flow. Linear Parameter-Varying (LPV) model was suggested where the model deals with a non-linear traffic flow system which contains autonomous and human-driven vehicles with different operating conditions. LPV provides the ability to control the nonlinear system which uses different linear controllers for different operating points. LPV model uses a scheduling variable to enable the controller based on the current operating point of the system [6]. This paper discusses the use of an MPC controller for an autonomous vehicle steering system and its implementation using MATLAB Simulink and an FPGA board. The implementation on FPGA is conducted using HDL coder.

This paper is organized as follows: in this first section, a review of the MPC formulation, previous work, and literature are presented. The second section describes the plant (the vehicle) for which the controller was implemented. Section three describes the simulation and implementations. Section four presents the obtained results and analyzes the implementation. Finally, the conclusions are provided and directions for future work are suggested.

2 MPC and Adaptive MPC Working Principles

In a control problem, basically, the goal of the controller is to calculate the input variables to the plant so the plant responds in a way that makes its output track the reference output. Figure 1 shows the standard control loop diagram.

2.1 Model Predictive Controller (MPC)

Model Predictive Control (MPC) uses a future prediction strategy in order to calculate the input. To ensure that the output of the plant follows the target reference output, the MPC controller uses what is called an optimizer. The prediction strategy is based on the use of a plant model (car model) by the MPC controller to simulate the car's path in the next P time steps, where P is the prediction horizon which represents the time, the MPC controller looks forward in the future to make the prediction. The Model Predictive Controller simulated different future scenarios in a systematic way, and here the optimizer comes to the picture by determining the best scenario which achieves the minimum error between the reference and the predicted trajectory. The minimum error corresponds to the minimum cost function, which means the scenario of the predicted trajectory with the minimum cost function provides the optimal solution. Figure 2 shows the traditional MPC controller, and Figure 3 shows a future prediction strategy, where each scenario represents a series of steering wheel movements in order to follow the reference trajectory, and as mentioned above the optimal scenario is the one which achieves the minimum cost function.

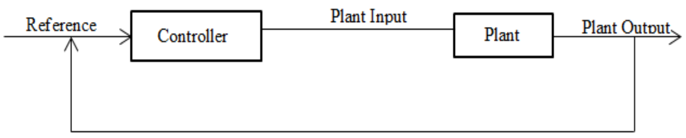

Figure 1
Standard Control Loop

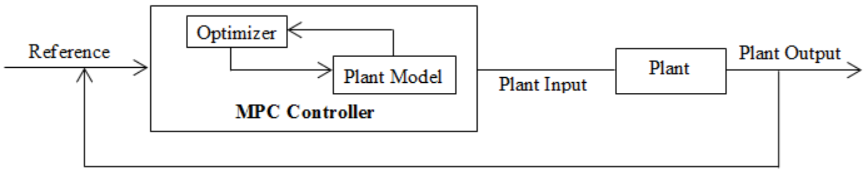

Figure 2
Traditional MPC control diagram

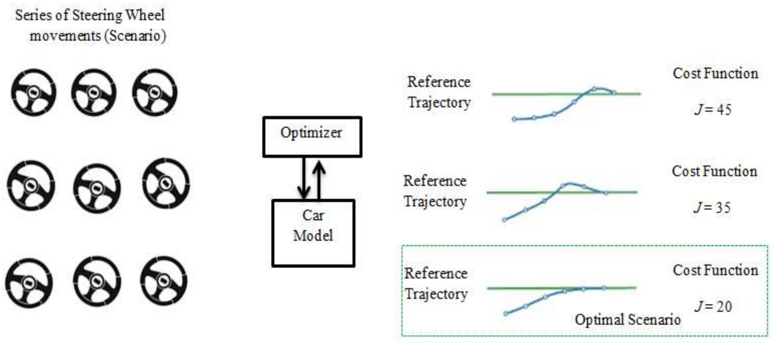

Figure 3
Future Prediction Strategy for Optimization Problem

The scenario with the minimum cost function J = 20 is the optimal solution, which achieves the optimal reference trajectory tracking.

The design presented in this article proposes that the new state of the car model can be measured, while in the case of the state model cannot be measured. The MPC controller uses the so-called “state estimator” to estimate the state of the system and feed it back to the controller. The MPC controller uses static Kalman Filter (KF) in order to update the controller states (plant model states, measurements noise model state and disturbance model state).

2.2 Adaptive Model Predictive Controller

The traditional MPC controller is unable to deal with the changing dynamics systems effectively since it uses a constant internal plant. When the system is nonlinear or its conditions vary with time, the accuracy will be negatively affected and the performance becomes unacceptable. To deal with these systems, an Adaptive MPC (AMPC) controller is used. AMPC controller handles the changes in operating conditions by providing a new linear model at each time step to achieve accurate prediction for the new conditions, as shown in Figure 4.

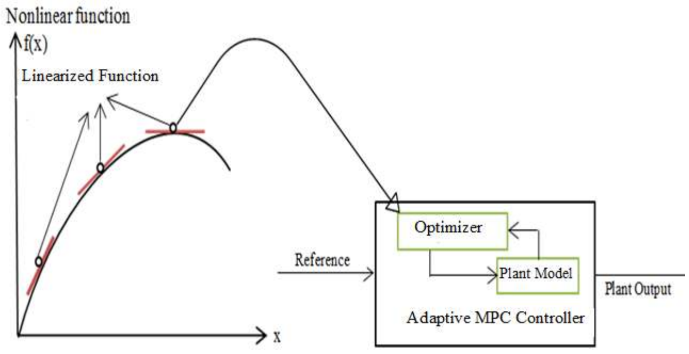

Figure 4
Adaptive MPC Controller [14]

The optimization problem in the Adaptive MPC controller remains the same, which means the same number of states and constraints for the varied operating conditions. The Adaptive MPC controller requires a discrete plant model, which means, the continuous-time state space needs to be converted to discrete-time (zero-order hold method). The Adaptive MPC Controller receives the updated discrete-time state space containing the following:

- A: n_x by n_x matrix signal, where n_x the number of plant model states.
- B: n_x by n_u matrix signal, where n_u the total number of plant inputs.
- DX: Vector signal of length n_x

$$DX = Ax_k + Bx_k - x_k \tag{1}$$

where DX is computed by equation (1), which provides the updated discrete-time state where u_k and x_k are respectively the inputs and the state values for the current time step k.

2.3 The optimization Problem

The MPC controller solves an online optimization problem, which is a Quadratic Problem (QP) for specific at each control interval. The optimization problem includes the followings:

Cost Function: also called objective function, it measures the controller performance, and the goal is to be minimized.

Constraints: It represents the soft and hard constraints which must satisfy the system conditions such as the physical bound.

To achieve the optimization, the MPC controller needs to calculate the control inputs driving the output of the plant that are very close to the desired reference. This process is performed in a systematic way by applying different scenarios and minimizing the cost function of the optimization problem. The cost function J of the autonomous vehicle's steering system can be formulated as:

$$\sum_{i=1}^{P} w_e e_{k+i}^2 \; \sum_{i=0}^{P-1} w_{\Delta u} \, \Delta u_{k+i}^2 \tag{2}$$

where w_e is the weight of the predicted error e_{k+1} and $w_{\Delta u}$ is the weight of the steering angle increments Δu_{k+1}. Cost function goals are to minimize both, the error between the predicted trajectory and the reference and the change in the steering angle between the consecutive time steps. The optimal solution corresponds to the smallest value of the cost function.

Decision: Modify the manipulated variables in order to achieve the minimization of the cost function and to satisfy the constraints.

The MPC controller computes the manipulated variable by solving the quadratic problem using a custom QP solver which in turn converts the linear optimization problem to the general form of the QP problem. Figure 5 shows the control algorithm of the Model Predictive Controller.

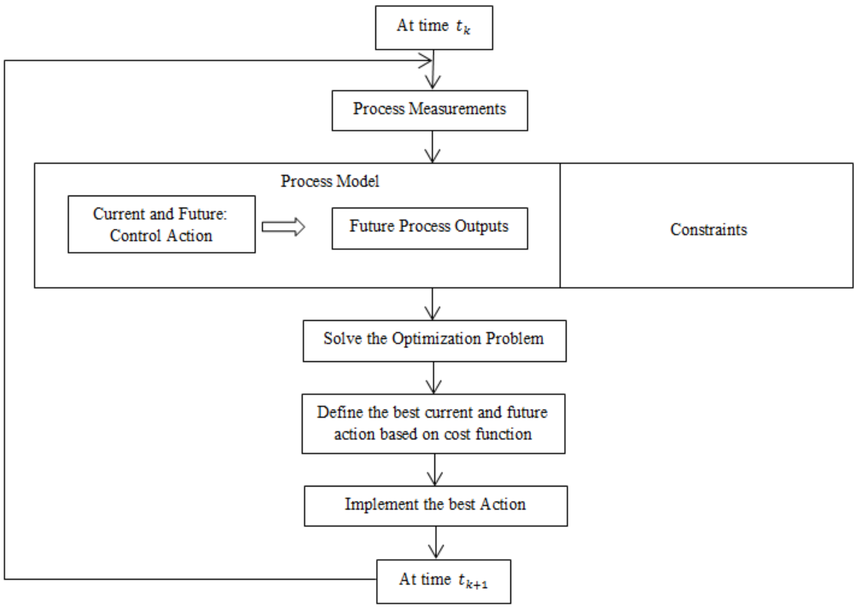
At time t_k
Process Measurements
Process Model
Current and Future: Control Action
Future Process Outputs
Constraints
Solve the Optimization Problem
Define the best current and future action based on cost function
Implement the best Action
At time t_{k+1}

Figure 5
MPC Control Algorithm

2.4 Model Predictive Controller Design Parameters

Designing the MPC controller takes into consideration the required constraints such as the steering angle limits. Figure 6 presents the main parameters and terms of the MPC controller, where the following nomenclature applies: k is the current sampling step and T_s the Control Time Step. Prediction horizon (P): number of time steps (the time on which the MPC controller looks forward to the future to make the prediction). Control Horizon (M): number of the possible control moves to time step $k+P$. The design parameters of the MPC controller are very important as this affects the performance and the computational complexity of solving the optimization problem. The choice of the design parameters should achieve the balance between the computational load and the performance. There are general recommendations, which can be taken into consideration for the parameters.

Sample time (T_S): determines the rate that the controller executes the control algorithm. In the case of Control Time Step T_s interval is too long, the controller will not be able to respond in time to the disturbance, which means that the performance will be negatively affected. On the other hand, if T_s is too short, the controller's response will be faster, but this causes a significant increase in computational load. The recommendation, in this case, is to choose T_s between 10 to 20 samples of the Rise Time T_r in an open-loop system, where T_r is the required time that the response takes to rise from 10 % to 90% of the steady-state as Figure 7 shows [15].

Prediction horizon (P): should be chosen in a way that covers the dynamic changes of the system and the recommendation are to choose P to have 20 to 30 of samples covering the open-loop transit system response [15], [18], [26] and [29].

Control Horizon (M): Only the two control moves have a significant impact on the response behavior, choosing a large control horizon will only increase the computation complexity, based on that, the recommendation is to choose M to be 10 to 20 of the prediction horizon. A small value of M provides stability while in contrast, large values reduce the robustness. It is recommended to choose M to be between 3-5 – as presented in [9], [15], [18], and [25].

For the model in this paper, the following strategy was used in order to choose the parameters which achieve satisfactory control performance: First, we initialized the parameters based on the recommendations above regarding the Sample Time, Prediction Horizon, and Control Horizon. Next step, is about tuning the parameters and then evaluating the MPC controller performance using the MPC Designer MATLAB toolbox until the optimal values provided the best control performance were determined. The weights of the inputs and outputs were determined using the MPC Designer by setting nonzero values to the inputs and outputs which need to track a reference value. Based on that, the weight equal is set to zero for the steering angle as it does not track a target. The weight of the

Lateral Position and Yaw angle were determined with nonzero values as the main objective is position tracing.

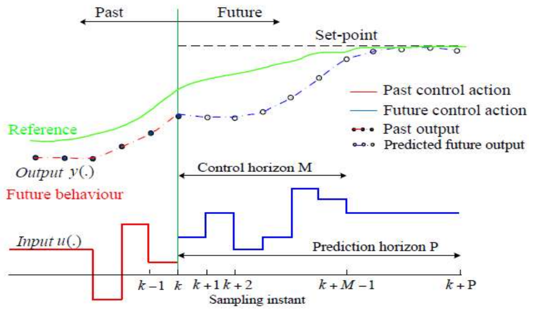

Figure 6

MPC schema for the main terms [27]

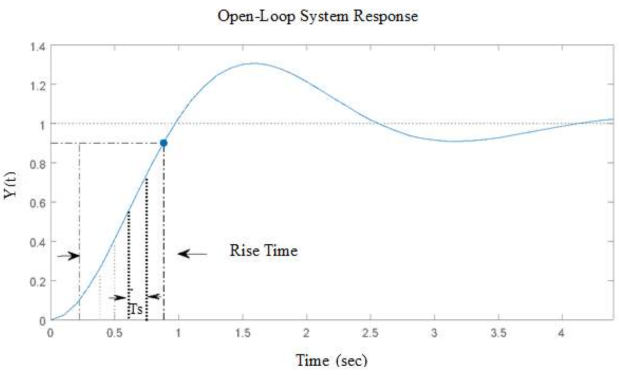

Figure 7

Control Time Step T_s and Rise Time T_r

3 The vehicle Model

MATLAB MPC designer application was used to design the controller that steers the vehicle autonomously. Figure 8 shows the global position of the vehicle in X and Y axes where (X, Y) are the vehicle's global position, v_y is the lateral velocity and v_x is the lateral longitudinal velocity. The parameters that need to be controlled are: Yaw angle Ψ and the front steering angle δ. The state-space of the model is given by the following equations:

$$\frac{d}{dt}\begin{pmatrix}\dot{y}\\ \psi\\ \dot{\psi}\end{pmatrix}=\begin{pmatrix}\frac{-2C_{af}+2C_{ar}}{mV_x} & 0 & -V_x-\frac{2C_{af}l_f-2C_{ar}l_r}{mV_x}\\ 0 & 0 & 1\\ \frac{2l_fC_{af}-2l_rC_{ar}}{I_zV_x} & 0 & \frac{-2{l_f}^2C_{af}+2{l_r}^2C_{ar}}{I_zV_x}\end{pmatrix}\begin{pmatrix}\dot{y}\\ \psi\\ \dot{\psi}\end{pmatrix}+\begin{pmatrix}\frac{2C_{af}}{m}\\ 0\\ \frac{2l_fC_{af}}{I_z}\end{pmatrix}\delta(2) \qquad (1)$$

$$\dot{y} = v_x\psi + v_y \qquad (2)$$

where v_x is longitudinal velocity at the center of gravity of the vehicle, m is the total mass of the vehicle, l_z is yaw moment of inertia of the vehicle, l_f and l_r are the longitudinal distance from the center of gravity to the front tires, C_{af} is cornering stiffness of tires and y is the lateral position.

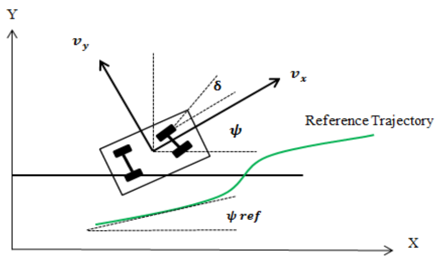

Figure 8
The global position of the vehicle

The MPC controller performs all the calculations using discrete-time state space. When a plant model is specified for the MPC controller, the following process needs to be performed [16]:

Conversion to state space: the model is converted to linear time invariant (LTI) state space model.

Discretization or resampling: in the case of difference sample time between the model and the MPC controller the following occurs:

- In the case of a continuous model, it must be converted to a discrete–time dynamic system model.
- In the case of the discrete model, the discrete-time dynamic system model is resampled in order to generate equivalent discrete–time model with a new sample Time T_S.

There are different ways to discretize a continuous model, in the proposed one, the continuous-time dynamic system model was discretized using zero–order hold on the inputs and sample time of T_S. This can be used also for resampling the discrete-time dynamic system model with new sample time T_S.

4 Design of the MPC Controller and HDL Code Generation

Based on the MPC control diagram the Simulink model was built. First, the required blocks (Plant model and Reference) were added to the workspace and linked to the MPC controller. The first input of the controller is the measured output and the second one is the reference trajectory, which was created using the Driving Scenario Designer Toolbox in MATLAB. As mentioned before, the MPC controller was designed using MPC Designer, where the internal plant model and the scenario are defined and the designing parameters such as sample time and control horizon were set using the strategy defined in section (2.4). In addition, the hard and soft constraints and their weights for the inputs and outputs such as the steering angle and the rate of change were set. In the case of an unchanging dynamics system, the input of the vehicle model is the output of the Model Predictive Controller (the steering angle) and the outputs are the lateral position and Yaw angle. Figure 9 presents the MPC controller model for linear systems (unchanging dynamics system). On the other hand, in the case of changing dynamics system, the longitudinal velocity is a second input for the vehicle model and the Adaptive MPC controller will use the plant mode output (State) to perform the new prediction for the updated model state. Figure 10 presents the Adaptive MPC controller model for nonlinear systems (changing dynamics system) with the Update Plant Model block.

Manual coding is time-consuming compared to the automatic code generation, which in turn lets the designers to focus on verification, validation and testing rather than programming. The model-based design generally provides an effective improvement in terms of system reliability and reduces the total project time up to 33% and the cost by 20% compared to the traditional methods (hand–written code) [23].

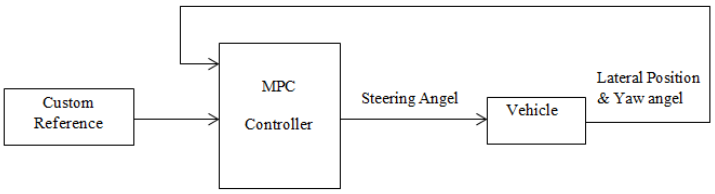

Figure 9
MPC controller model for linear system (Constant longitudinal velocity)

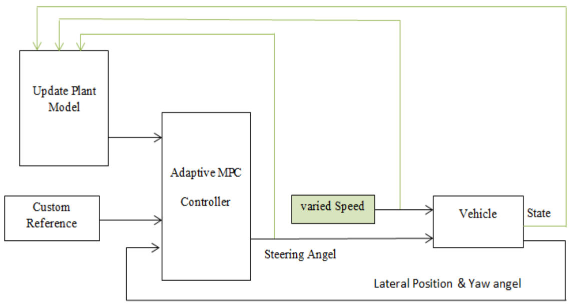

Figure 10
Adaptive MPC controller model for nonlinear system (varied longitudinal velocity)

The floating-point model needs to be converted to fixed point in order to reduce the hardware resources [19]. The steering system was designed and simulated using MATLAB Simulink and implemented on SoC (System on Chip) target using embedded coder and HDL coder. The working methodology is presented in Figure 11. First, the MPC controller model was created and the parameters were determined in MATLAB (see Table 1), followed by the HDL coder model and functional verification. Intellectual Property (IP) was created by Vivado. The MPC controller project was created and the MPC IP was connected to the Processing System (PS) through AXI interface. Figure 12 shows the block design of the MPC system.

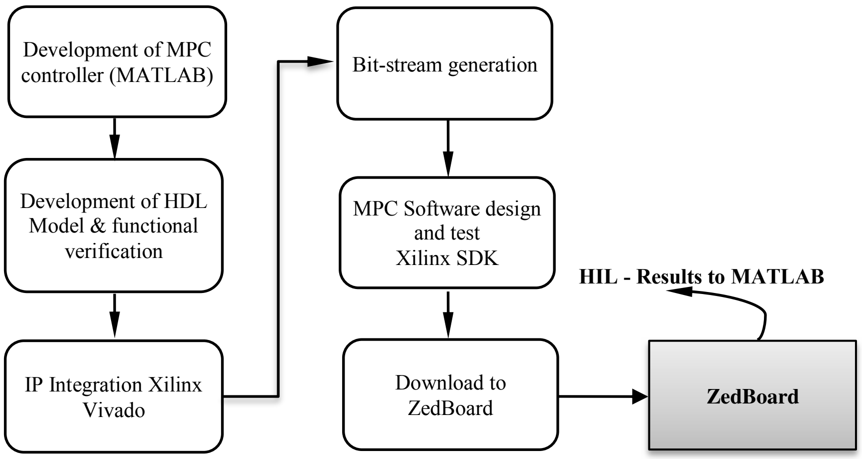
Development of MPC controller (MATLAB)
Development of HDL Model & functional verification
IP Integration Xilinx Vivado
Bit-stream generation
MPC Software design and test Xilinx SDK
Download to ZedBoard
HIL - Results to MATLAB
ZedBoard

Figure 11
The design workflow of the proposed solution [2]

Table 1
Values of the main MPC controller parameters and constraints

MPC Parameters	
Parameter	**Value**
Sample Time Ts	0.1 seconds
Prediction Horizon (P)	10 seconds
Control Horizon (M)	3 seconds
Constraints	
Steering Angle	[-0.5 - 0.5] rad
Steering Angle (changing rate)	[-0.26 - 0.26] rad

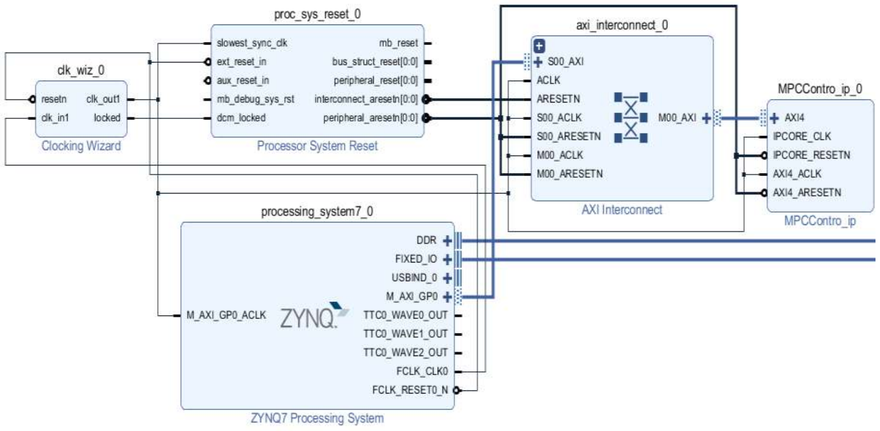

Figure 12
Vivado Block Design

The next step of the development (see Figure 11) was the bit-stream generation and export to the software development system (Xilinx SDK). The last step of the development was the software design and test. The generated project in Xilinx SDK together with the bit-stream downloaded and the target FPGA was programmed. In MATLAB Simulink the MPC model and MPC hardware system were tested and checked with Hardware In the Loop (HIL) simulation. The results are presented in the next section.

5 Simulation and Implementations Results

5.1 MATLAB Simulink Implementations

The steering system model was tested using MATLAB Simulink for both MPC and Adaptive MPC. Figure 13 shows the performance of MPC controller at a constant longitudinal velocity, and Figure 14 shows its performance at varied longitudinal velocity. The obtained results in Figure 13 and Figure 14 show that the MPC controller achieved satisfactory performance for the constant operating conditions, while it failed to handle the system with changing longitudinal velocity. Figure 15 shows the performance of the Adaptive MPC controller for the changing dynamic system (varied longitudinal velocity). Results demonstrate that using the Adaptive MPC controller for the changing dynamics system yields good performance in terms of tracking the reference (lateral position and yaw angle).

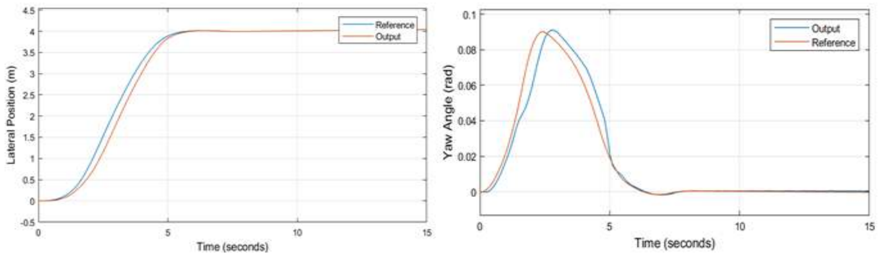

Figure 13
MPC controller performance at constant velocity

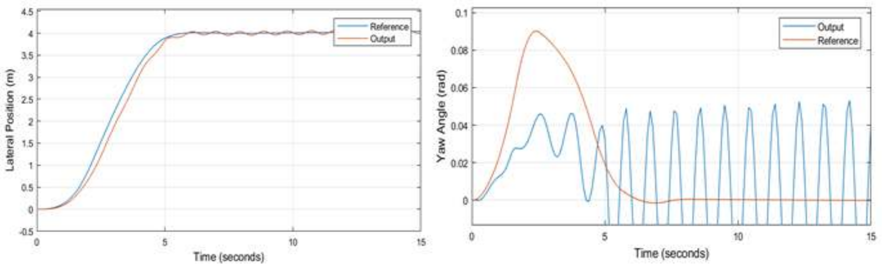

Figure 14
MPC controller performance at varied velocity

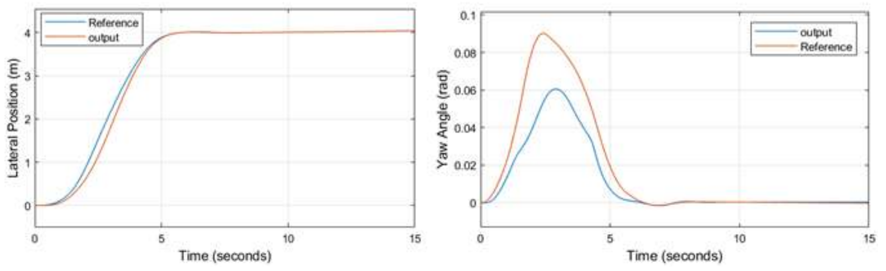

Figure 15
Adaptive MPC performance at varied velocity

5.2 FPGA Implementations

Both models (MPC and Adaptive MPC controller) were implemented on FPGA and the results were compared with the results obtained using MATLAB Simulink. The experiments showed slight differences in terms of performance between the implementations (Simulink and FPGA). Figure 16 and Figure 17 show the performance of the MPC controller at constant longitudinal velocity, and the performance of the Adaptive MPC controller at varied longitudinal velocity, respectively. Figure 18 and Figure 19 clearly show the difference in performance between the two controllers' implementation.

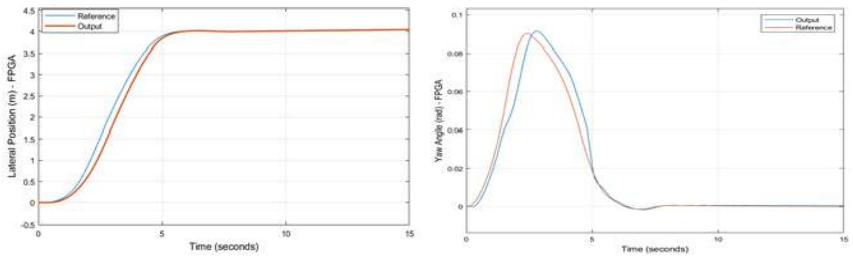

Figure 16
MPC controller performance at constant longitudinal velocity (FPGA)

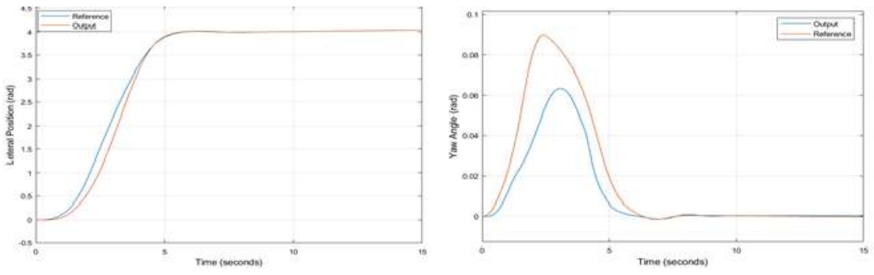

Figure 17
Adaptive MPC controller performance at varied longitudinal velocity (FPGA)

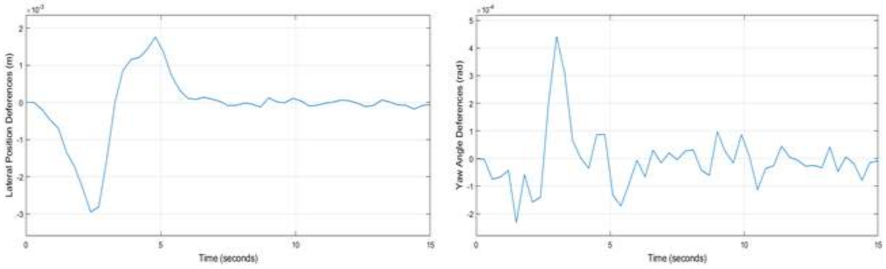

Figure 18

MPC implementation using Simulink and FPGA: Performance compression

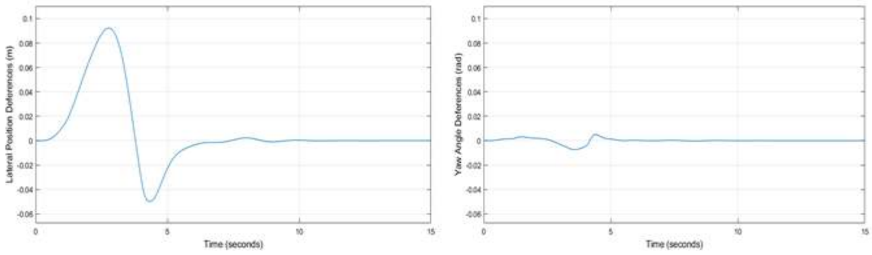

Figure 19

Adaptive MPC implementation using Simulink and FPGA: Performance compression

The implementations of MPC and Adaptive MPC controllers on FPGA were analyzed also in terms of resource utilization and power consumption using three different strategies for implementation to achieve the optimization as Table 2 and Table 3 show. In general, the implementations involve Logical optimization, placement of logic cells, and routing the connections between cells [28]. Implementation "Defaults strategy" balances runtime with trying to achieve timing closure. "Performance_ExplorePostRoutePhysOpt" strategy uses multiple algorithms for optimization, placement, and routing in order to get potentially better results. In "Flow_RuntimeOptimized" strategy, each implementation step trades design performance for a better run time [28].

Table 2

Resource utilization using different strategies

Defaults strategy					
Resource	**Utilization**		**Available**	**Utilization %**	
	MPC	Adaptive MPC	Adaptive MPC - MPC	MPC	Adaptive MPC
LUT	204	208	53200	0.38	0.39
FF	361	361	106400	0.34	0.34
BUFG	3	3	32	9.38	9.38

Performance_ExplorePostRoutePhysOpt strategy					
LUT	181	184	53200	0.34	0.35
FF	329	329	106400	0.31	0.31
BUFG	3	3	32	9.38	9.38
Flow_RuntimeOptimized strategy					
LUT	177	231	53200	0.33	0.43
FF	329	361	106400	0.31	0.34
BUFG	3	3	32	9.38	9.38

Table 3

Power consumption – different implementation strategies

Name	**Strategy**	**Total Power (W)**	
		MPC	Adaptive MPC
Impl_1	Implementation Defaults	1.791	1.791
Impl_2	Performance_ExplorePostRoutePhysOpt	1.792	1.792
Impl_3	Flow_RuntimeOptimized	1.793	1.791

Table 4

Power Consumption on chip - Summary

	Power Consumption	**Power on Chip**	
Dynamic	91%	Clocks	Less than 1%
		Signals	Less than 1%
		Logic	Less than 1%
		MMCM	6%
		PS7	91%
Static	9%	PL Static	100%

The results in Table 2 show that the implementation of the MPC controller on FPGA using the "defaults" strategy has the highest resource utilization, whereas the "Flow_RuntimeOptimized" strategy achieved the lowest resource utilization, where the utilization of LUTs (Lookup Tables) and FF (Flip-Flop) were reduced by 13.2% and 8.86% respectively. For BUFG (Global Buffer) there is no change. On the other hand. the implementation of MPC controller using "Performance_ExplorePostRoutePhysOpt" strategy achieved the lowest resource utilization. Table 3 shows that the power consumption for all applied strategies is almost the same.

Table 4 shows that 91% of the total power was used by the Processing System (PS), whereas only 9% was used by Programmable logic (PL) and only 6% of MMCM (Mixed-Mode Clock Manager) were used for both MPC and AMPC implementations.

Conclusions

This paper discussed the implementations of MPC and adaptive MPC controllers to control an autonomous vehicle steering system. The implementations were performed for both constant and changing dynamics systems. The models were implemented on FPGA using MATLAB HDL coder and different strategies were adopted to optimize resource utilization. The results showed that the MPC controller provides a satisfactory control for a constant dynamics system, but it couldn't handle operating conditions that are changing, while adaptive MPC provides good control for changing dynamics systems. In addition to analyzing the performance of the controllers, the implementations were discussed in terms of resource utilization and power consumption using different strategies.

The results showed a very slight improvement regarding the total power consumption. Based on the findings of this study, in future work, the implementations of MPC and adaptive MPC controller will be performed using System Generator in order to improve the power consumption and results will be compared with the results obtained in this paper.

Acknowledgement

This research was supported by the European Union and the Hungarian State, co-financed by the European Regional Development Fund in the framework of GINOP-2.3.4-15-2016-00004 project, aimed to promote the cooperation between the higher education and the industry.

References

[1] V. Adetola, M. Guay, Robust adaptive MPC for constrained uncertain nonlinear systems. International Journal of Adaptive Control and Signal Processing, 2011, pp. 155-167

[2] H. Borhan, A. Vahidi, A. Phillips, M. Kuang, I. Kolmanovsky, S. Di Cairano, MPC-Based Energy Management of a Power-Split Hybrid Electric Vehicle. IEEE Transactions on Control Systems Technology 20, 2012, pp. 593-603

[3] L. Crockett, D. Northcote, C. Ramsay, F. Robinson, R. Stewart, Exploring Zynq MPSoC: With PYNQ and Machine Learning Applications, https://www.zynq-mpsoc-book.com, UK, 2019, pp.

[4] S. Di Cairano, I. Kolmanovsky, Real-time optimization and model predictive control for aerospace and automotive applications. In: 2018 Annual American Control Conference (ACC), USA, 2018, pp. 2392-2409

[5] S. Di Cairano, I. Kolmanovsky, Automotive applications of model predictive control, Handbook of Model Predictive Control. Control Engineering, 2019, pp. 493-527

[6] Gy. Eigner, Control of physiological systems through linear parameter varying framework. Acta Polytechnica Hungarica, Vol. 14, No. 6, 2017, pp. 185-212

[7] P. Falcone, F. Borrelli, H. Tseng, J .Asgari, D. Hrovat, A Hierarchical Model Predictive Control Framework for Autonomous Ground Vehicles. American Control Conference, 2008, pp. 3719-3724

[8] P. Falcone, H. Tseng, F. Borrelli, J .Asgari, D. Hrovat, MPC-based yaw and lateral stabilization via active front steering and braking. Vehicle System Dynamics: International Journal of Vehicle Mechanics and Mobility, 2008, pp. 611-628

[9] J. Garriga and M. Soroush, Model predictive control tuning methods: A review. *Ind. Eng. Chem. Res.*, Vol. 49, No. 8, 2010, pp. 3505-3515

[10] T. Haidegger, L. Kovacs, S. Preitl, R. E. Precup, B. Benyo, Z. Benyo, Controller Design Solutions for Long Distance Telesurgical Applications. International Journal of Artificial Intelligence, Vol. 6, No. S11, 2011, pp. 48-71

[11] M. Lau, S. Yue, K. Ling, J. Maciejowski, A Comparison of Interior Point and Active Set Methods for FPGA Implementation of Model Predictive Control. Proceedings of European Control Conference, 2009. pp. 156-161

[12] K. Ling, B. Wu, J. Maciejowski, Embedded Model Predictive Control (MPC) using a FPGA. The International Federation of Automatic Control, 2008, pp. 1930-1935

[13] K. Ling, S. Yue, J. Maciejowski, A FPGA Implementation of Model Predictive Control. American Control Conference, 2006, pp. 15250-15255

[14] MathWorks ***, I. 2018. Linearize Nonlinear Models, URL: https://www.mathworks.com/help/slcontrol/ug/linearizing-nonlinear-models.html#responsive_offcanvas, Last accessed 16 March 2020

[15] MathWorks ***, I. 2018. Choose Sample Time and Horizons, URL: https://www.mathworks.com/help/releases/R2018a/mpc/ug/choosing-sample-time-and-horizons.html?s_eid=PSM_15028, Last accessed 16 March 2020

[16] MathWorks ***, I. 2018. MPC Modelling, URL: https://www.mathworks.com/help/mpc/gs/mpc-modeling.html, Last accessed 25 March 2020

[17] B. Németh, G. Péter, LPV design for the control of heterogeneous traffic flow with autonomous vehicles. Acta Polytechnica Hungarica, Vol. 16, No. 7, 2019, pp. 233-246

[18] Q. T. Nguyen, V. Veselý, D. Rosinová, Design of robust model predictive controller with input constraints. International Journal of Systems Science, Vol. 44, No. 5, 2013, pp. 896-907

[19] N. Othman, F. Mahmud, A. K. Mahamad, M. H. Jabbar, N. A Adon, Cardiac Excitation Modeling: HDL Coder Optimization towards FPGA stand-alone Implementation, In: 2014 IEEE International Conference on Control System, Computing and Engineering, 28-30 November, 2014, pp. 507-511

[20] T. Haidegger, L. Kovács, R. E. Precup, S. Preitl, B. Benyó, Z. Benyó, Cascade Control for Telerobotic Systems Serving Space Medicine. IFAC World Congress, Vol. 44, No. 1, 2011, pp. 3759-3764

[21] C. F. D. Saragih, F. M. T. R. Kinasih, C.Machhbub, P. H. Rusmin, A. S. Rohman, Visual Servo Application Using Model Predictive Control (MPC) Method on Pan-tilt Camera Platform. 6th International Conference on Instrumentation, Control, and Automation (ICA), August 2019, pp. 1-7

[22] M. Schetzen, Linear Time-Invariant Systems. John Wiley & Sons, New York, 2003

[23] Y. Siwakoti, G. Town, Design of FPGA-Controlled Power Electronics and Drives Using MATLAB Simulink. IEEE ECCE Asia Down under conference, 2013, pp. 571-577

[24] P. Skarin, W. Tärneberg, K. E. Årzen, M. Kihl, Towards Mission-Critical at the Edge and Over 5G. in 2018 IEEE International Conference on Edge Computing (EDGE), 2018, pp. 50-57

[25] S. E. Tuna, M. J. Messina and A. R. Teel, Shorter horizons for model predictive control. Proceeding of the 2006 American Control Conference, 2006, pp. 863-868

[26] K. Worthmann, Estimates of the prediction horizon length in MPC: A numerical case study, Proc. IFAC Conf. Nonlinear Model Predictive Control, 2012, pp. 232-237

[27] Y. Xiaoliang, L. Guorong, L. Anping, L. Van Dai, A Predictive Power Control Strategy for DFIGs Based on a Wind Energy Converter System, Energies, Vol. 10, No. 8, 1098, 2017, pp. 2-24

[28] Xilinx ***, Vivado design suite user guide: Implementation. UG904, v2016.2, 2016

[29] A. S. Yamashita, A. C. Zanin, D. Odloak, Tuning of model predictive control with multi-objective optimization. *Brazilian J. Chem. Eng.*, Vol. 33, No. 2, 2016, pp. 333-346