Embedded Model Predictive Controller on Low-Cost Low-End Microcontroller for Electrical Drives

Saket Adhau[1], Siddesh Dani[2], Deepak Ingole[3], and Dayaram Sonawane[1]

***Abstract*— It is very well-known that the implementation of Model Predictive Controller (MPC) on embedded platforms is challenging due to the computational complexities associated while solving an optimization problem. Although, there are many efficient embedded implementations existing by now, but for faster, more dynamic and non-linear control applications, there is no cost-effective and memory efficient embedded solutions. In this paper, we show the implementation of embedded explicit MPC for a motor speed control application on a low-cost $8\,\mathrm{bit}$ PIC 18 series microcontroller which costs only $\$5$. The offset-free explicit MPC is designed for reference tracking, constraints handling, and disturbance rejection. The developed control law is exported to low-level C code and utilized in HIL co-simulations. We present the results of memory demand and control performance under various operating scenarios. The presented results show that the developed embedded MPC utilize about 40$\%$ of RAM and 92$\%$ of ROM for prediction horizon up to 3 samples. The performance of developed MPC is compared with the conventional PI controller. Overall results show that the presented approach is cost-effective, portable, and gives better performance than the PI controller.**

***Index Terms*— motor control, model predictive control, disturbance modeling, microcontroller, optimization.**

I. INTRODUCTION

Looking at the performance of Model Predictive Controller (MPC), along with the better constraint handling, offset-free reference tracking and optimal control actions prove the best choice in many process industries and extended applications. There are many different ways to implement MPC on embedded platforms for standalone applications. MPC has also been implemented on System on Chip (SoC) by [1] which presents a basic framework and hardware architecture required for real-time MPC. Authors in [2] have developed MPC for micro-chemical systems using Motorola's MPC555 which has high speed $32\,\mathrm{bit}$ CPU and special $64\,\mathrm{bit}$ floating-point unit designed to accelerate advanced algorithms for complex applications. Many different embedded architectures for linear MPC have been presented in [3]. Field Programmable Gate Array (FPGA) is another hardware tool which has been extensively used for implementing MPC [4]. FPGA requires more coding efforts and are complex in prototyping. Also, they consume a lot more power than any other [5]. So, research is being carried out to implement MPC on microcontrollers which are less expensive and consume less power than FPGAs. On the similar line, Generalized Predictive Control (GPC) on an embedded platform (STM32) has been proposed by the authors in [5]. The authors in [6] have implemented constrained MPC for the first order and second order plant on the STM32 kit and have compared the performance with anti-windup Proportional-Integral-Derivative (PID).

Based on the optimization problem-solving approach MPC can be categorized in two ways, namely on-line MPC and off-line or Explicit MPC (EMPC). On-line MPC is computationally burdening on the processor and require a longer time for solving quadratic problems. Therefore, it requires processor working at higher clock speeds. On the other hand, explicit MPC comprised of some data structure in the form of Look-Up Tables (LUTs) and point location algorithm which efficiently searches through for current optimal MPC solution. Hence, it is computationally less complex and therefore less computationally burdening on the processor [7]. On the downside, the memory requirement for EMPC is high as compared to on-line MPC and it increases with the horizons. Also, there are many EMPC solutions available on embedded platforms. Authors in [8], [9] has proposed an FPGA-based EMPC scheme using Binary Search Tree (BST) to find the optimal rate of anesthesia drug infusion in the patient. The authors in [10] have presented a real-time application of EMPC for active vibration compression using a $32\,\mathrm{bit}$ ARM Cortex M4F (STM32F407VGT6).

Motivated by these research ideas and our previous work on MATLAB® implementation of on-line MPC [11]. This paper attempts to implement EMPC solution with disturbance modeling on low-cost microcontroller for speed control of DC motor. The sole reason for selecting this low-cost microcontroller is to prove that the explicit MPC can be implemented on low-end microcontroller for dedicated standalone applications. An offset-free explicit MPC is designed for the Multi-Input Single-Output (MISO) motor model using disturbance modeling approach. Furthermore, a low-level C code of the explicit MPC is exported and deployed on the microcontroller which is then used to control the desired speed reference using Hardware-in-the-Loop (HIL) co-simulation. The obtained results of the designed explicit MPC are presented for reference tracking, disturbance rejection, and constraint handling along with the results of PI controller.

II. DC MOTOR MODELING

In this section, a mathematical model of permanent magnet DC motor is presented. The model is derived using torque

[1] Department of Instrumentation and Control Engineering, College of Engineering, Pune, India, `adhauss17.instru@coep.ac.in`, `dns.instru@coep.ac.in`.
[2] Renu Electronics Private Limited, Pune, India, `danisk15.instru@coep.ac.in`.
[3] University of Lyon, IFSTTAR, ENTPE, Lyon, France, `deepak.ingole@ifsttar.fr`.

and electrical characteristics described in [12, Chapter 2]. The electrical circuit of the motor is shown in Fig. 1. A voltage source (V_a) across the armature coil, induced voltage or back electromotive force (emf) (V_m) which opposes the voltage source and an inductance (L_a) in series with a resistance (R_a) forms the closed circuit. The rotation of electrical coil through fixed flux lines of permanent magnets generates the back emf. Differential equations for the electrical circuit

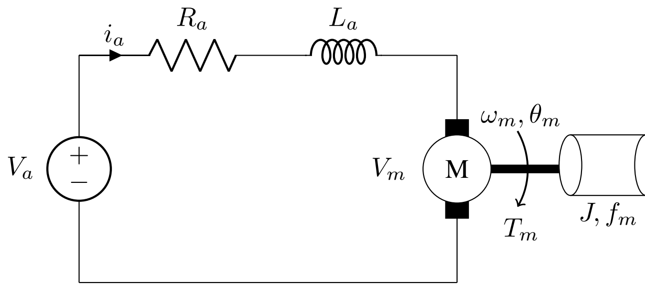

Fig. 1. Electrical equivalent circuit of DC motor.

shown in Fig. 1 are derived using Kirchoff's voltage law. The sum of all voltages around a loop can be given using Ohm's and Kirchoff's law

$$L_a \frac{di_a}{dt} + R_a i_a + V_m = V_a, \tag{1}$$

where $\frac{di_a}{dt}$ is the change in current through the coil with respect to time and i_a is the armature current. The back emf is obtained by

$$V_m = k_m \omega_m, \tag{2}$$

where ω_m is the rotational speed of the rotor and k_m is the back emf constant described by the reluctance of the iron core of armature, flux density of permanent magnets and the number of turns in the armature winding.

The applied voltages at stator and rotor causes the motor to exerts a torque. This torque acting on the mechanical shaft is characterized by the viscous friction coefficient f_m and rotor inertia J. If external disturbances are acting on the shaft of the motor categorized as load torque ; then, T_l is load torque and $T_m = k_m i_a$ is emf torque, the following equation can be written for the total torque of motor as

$$J\frac{d\omega_m}{dt} + f_m \omega_m = k_m i_a - T_l. \tag{3}$$

Applying the Laplace transform to motor model (1) and (3), in terms of the Laplace variable s we get

$$(L_a s + R_a) i_a(s) + k_m s \omega_m(s) = V_a(s), \tag{4}$$
$$s(Js + f_m)\omega_m(s) = k_m i_a(s). \tag{5}$$

Considering the armature voltage ($V_a(s)$) as the input variable, the rotational speed ($\omega_m(s)$) as the output variable and by eliminating $i_a(s)$ from (4) and (5), we arrive at the following open-loop transfer function,

$$G_{\omega_m V_a}(s) = \frac{k_m}{(R_a + sL_a)(Js + f_m) + k_m^2}. \tag{6}$$

For linearizing the system and as $L_a << R_a$, L_a term in (6) is neglected. We get the new transfer function as,

$$G_{\omega_m V_a}(s) = \frac{k_m}{R_a(Js + f_m) + k_m^2}. \tag{7}$$

The viscous friction coefficient (f) of motor can be stated as

$$f = f_m + k_m^2/R_a. \tag{8}$$

Further simplification gives,

$$G_{\omega_m V_a}(s) = \frac{K}{(\tau s + 1)}, \tag{9}$$

where K is steady state gain $\frac{k_m}{R_a f}$ and τ is the time constant of the system described as $\frac{J}{f}$. Similarly the transfer function of the angular position (θ_m) can be obtained by integrating angular speed i.e. multiplying $\frac{1}{s}$ to (7) as

$$G_{\theta_m V_a}(s) = \frac{K}{s(\tau s + 1)}. \tag{10}$$

Here, angular position and angular velocity as considered as the states and armature voltage as the input, the equation in state space form becomes,

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{11a}$$
$$y(t) = Cx(t) + Du(t), \tag{11b}$$

where system matrices A, B, C and D are given as

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{1}{\tau} \end{bmatrix}, B = \begin{bmatrix} 0 \\ \frac{K}{\tau} \end{bmatrix}, C = \begin{bmatrix} 0 \\ 1 \end{bmatrix}^T, D = 0.$$

Table I shows the system parameters for simulation of the model. In the next, an overview of designed control algorithms is given.

TABLE I
SYSTEM PARAMETERS.

Notation	Description	Value	Unit
k_m	Torque constant	8.32×10^{-4}	Nm/A
J	Moment of inertia	2.45×10^{-7}	kg.m^2
f_m	Viscous friction constant	3.10×10^{-5}	Nm/rad/s
R_a	Armature resistance	4.1	Ω
L_a	Armature inductance	2.27	mH

III. Proportional Integral (PI) Controller

PI controller algorithm consists of two blocks namely, the Proportional and Integral mode. The simplest form of the PI controller in continuous-time is given by

$$u_t = k_p e_t + k_i \int_0^t e_\tau d\tau, \tag{12}$$

where u is the control input, e is the error between reference and output, and k_p and k_i are the proportional and integral gain, respectively. The integral action removes the offset in the error and the proportional action is related to the increase of the control variable when the control error is large. The discrete time equation is:

$$u_k = k_p e_k + k_i T_s \sum_{j=0}^{k} e_j, \tag{13}$$

where T_s is the sampling time of the system.

IV. Offset-Free Explicit MPC

A. Plant Model

Consider a discrete-time version of the Linear-Time Invariant (LTI) system in (11),

$$x(t+T_s) = Ax(t) + Bu(t), \tag{14a}$$
$$y(t) = Cx(t) + Du(t), \tag{14b}$$

where $x(t) \in \mathbb{R}^n$ is the system state vector, $u(t) \in \mathbb{R}^l$ is the system input vector and $y(t) \in \mathbb{R}^m$ is the system output vector, moreover, $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times l}$, $C \in \mathbb{R}^{m\times n}$ and $D \in \mathbb{R}^{m\times l}$ are system matrices.

B. Disturbance Modeling

The objective is to design an explicit model predictive controller based on linear system model (11) in order to have measured output $y(t)$ track the desired reference $r(t)$ with zero steady-state error in presence of the plant-model mismatch and/or un-known disturbances. To achieve this objective, the plant model (14) is augmented with a disturbance vector $d(t) \in \mathbb{R}^p$ [13], [14, Chapter 12] as shown below in state space form

$$\underbrace{\begin{bmatrix} x(t+T_s) \\ d(t+T_s) \end{bmatrix}}_{x_e(t+T_s)} = \underbrace{\begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix}}_{A_e} \underbrace{\begin{bmatrix} x(t) \\ d(t) \end{bmatrix}}_{x_e(t)} + \underbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}_{B_e} u(t), \tag{15a}$$

$$y_e(t) = \underbrace{\begin{bmatrix} C & C_d \end{bmatrix}}_{C_e} \underbrace{\begin{bmatrix} x(t) \\ d(t) \end{bmatrix}}_{x_e(t)} + D_e u(t), \tag{15b}$$

where $B_d \in \mathbb{R}^{n\times p}$, $C_d \in \mathbb{R}^{m\times p}$ are the disturbance model matrices and dimensions of matrices 0 and I are $m \times l$ and $m \times m$ respectively. The subscript 'e' is the extended version of the combined disturbance and state.

C. State and Disturbance Estimation

Extended state x_e is estimated from the plant measurement by designing a Luenberger observer for augmented system (15) as follows,

$$\hat{x}_e(t+T_s) = A_e\hat{x}_e(t) + B_e u(t) + L_e(y(t) - \hat{y}_e(t)), \tag{16a}$$

$$\hat{y}_e(t) = C_e\hat{x}_e(t) + D_e u(t), \tag{16b}$$

where $L_e = \begin{bmatrix} L_x & L_d \end{bmatrix}^T$ is the filter gain matrices for the state (of dimension $n \times m$) and the disturbance (of dimension $p \times m$), respectively and can be obtained by pole placement.

D. MPC Formulation

In MPC, the control action is obtained by solving a Constrained Finite Time Optimal Control (CFTOC) problem for the current state ($\hat{x}_e(t)$) of the plant at each sampling time (t). The sequence of optimal control inputs ($U^\star = \{u_0^\star, \ldots, u_{N-1}^\star\}$) is computed for a predicted evolution of the system model over a finite horizon (or prediction horizon (N)). However, only the first element of the control sequence ($u_0^\star$) is applied and the current state ($\hat{x}_e(t)$) of the system is then measured again at the next sampling time ($t+1$). This so-called Receding Horizon Controller (RHC) which introduces feedback to the system, thereby allowing for the compensation of potential modeling errors or disturbances acting on the system [14, Chapter 12].

Using LTI system model in (14) and disturbance observer in (16) the MPC problem is designed as follows

$$\min_U \sum_{k=0}^{N-1} (y_k - r_k)^T Q (y_k - r_k) + \Delta u_k^T R \Delta u_k \tag{17a}$$
$$\text{s.t.} \quad x_{k+1} = Ax_k + Bu_k + B_d d_k, \tag{17b}$$
$$d_{k+1} = d_k, \tag{17c}$$
$$y_k = Cx_k + Du_k + C_d d_k, \tag{17d}$$
$$\Delta u_k = u_k - u_{k-1}, \tag{17e}$$
$$u_{\min} \leq u_k \leq u_{\max}, \tag{17f}$$
$$u_{-1} = 0, \tag{17g}$$
$$x_0 = \hat{x}(t), \tag{17h}$$
$$d_0 = \hat{d}(t), \tag{17i}$$
$$\forall k \in \{k = 0, \ldots, N-1\}, \tag{17j}$$

where $Q \in \mathbb{R}^{n\times n}$ and $R \in \mathbb{R}^{l\times l}$ are the weighting matrices, with condition $Q \succeq 0$ to be positive semi-definite, and $R \succ 0$ to be positive definite. We denote N as the prediction horizon, x_{k+1} as the vector of predicted states at sample time k, $U = \{u_0, \ldots, u_{N-1}\}$ as the sequence of control actions, r_k is the output reference trajectory to be tracked, and x_0, d_0, and u_{-1} are the given initial conditions.

By solving (17) with a given initial conditions, the optimization yields open loop optimal input sequence $U^\star = \{u_0^\star, \ldots, u_{N-1}^\star\}$ from which only the first control action, i.e., $u_0^\star$ is applied to the plant and again a CFTOC problem (17) is solved at next time instance $k+1$.

E. Explicit MPC

In MPC, an optimization problem (17) needs to be solved at each time instance. Such an optimization problem can be formulated as a multi-parametric Quadratic Programming (mp-QP) problem

$$\min_U \left\{ U^T H U + \tilde{x}_0^T F U \right\} + \tilde{x}_0^T Y \tilde{x}_0, \tag{18a}$$
$$\text{s.t.} \quad GU \leq w + W\tilde{x}_0, \tag{18b}$$

where $\tilde{x}_0 = [\hat{x}_e^T(t) \; u^T(t-1) \; r^T(t)]^T$ is the vector of initial conditions and by denoting q as a number of inequalities, matrices $H \in \mathbb{R}^{l.N\times l.N}$, $F \in \mathbb{R}^{(n+p)\times(l.N)}$, $Y \in$

$\mathbb{R}^{(n+p)\times(n+p)}, G \in \mathbb{R}^{q\times l.N}, w \in \mathbb{R}^q, W \in \mathbb{R}^{q\times n}$ can be obtained by weighting matrices Q and R [15].

The optimal solution $U^\star$ is a piecewise affine function of the initial condition, which can be computed off-line by solving mp-QP problem [15]. This mp-QP solution can be evaluated using MATLAB® based Multi-Parametric Toolbox (MPT) [16], [17]. Once the mp-QP problem (18) is solved off-line and stored locally in LUTs, explicit MPC uses the obtained optimal solution in a receding horizon fashion in which $U^\star = \kappa(\tilde{x}_0)$ is a continuous PWA function mapped over a polyhedral partition

$$\kappa(\tilde{x}_0) = \begin{cases} F_1\tilde{x}_0 + g_1 & \text{if } \tilde{x}_0 \in \mathcal{R}_1 \\ & \vdots \\ F_M\tilde{x}_0 + g_M & \text{if } \tilde{x}_0 \in \mathcal{R}_M \end{cases} \tag{19}$$

where $\mathcal{R}_i = \{\tilde{x} \in \mathbb{R}^{(n+p)} \mid Z_i\tilde{x} \leq z_i\} \quad \forall i = 1,\ldots,M$ are the polyhedral regions and $F_i \in \mathbb{R}^{l\times(n+p)}$, $g_i \in \mathbb{R}^l$ are optimal gains. Also, M states the total regions, h_i is a number of half-spaces in a polyhedral set and $Z_i \in \mathbb{R}^{h_i\times(n+p)}$, $z_i \in \mathbb{R}^{h_i}$ are the matrices which forms a polyhedral set. The advantage of explicit form of controller as shown in (19) is, the computation is reduced to only search algorithm consisting of addition and multiplications to find the optimal control inputs .

F. Evaluation of PWA Function

In the on-line phase of explicit MPC, the task is to identify an index of polyhedral region in which the current state $\tilde{x}_0$ is found. Once an index i of the corresponding region is identified, then the optimal control action can be calculated using (19). There exists numerous point location algorithms in literature and implemented in MPT [16], [17]. The simplest algorithm is sequential search, which checks constraint satisfaction $Z_i\tilde{x}_0 \leq z_i$ for all regions $i = 1,\ldots,M$ one after another until the region containing current state $\tilde{x}_0$ is found [8].

V. Embedded Realization of EMPC

The robust and scalable customized data packet is developed for USART to establish communication between microcontroller and MATLAB® for HIL co-simulation. Fig. 2 shows, the overview of HIL set-up with microcontroller and MATLAB/SIMULINK for the DC motor speed control using explicit MPC.

A. Construction of Explicit MPC

A combined (motor and disturbance model) discrete time LTI model (15) is considered to design an explicit MPC with 4 states (x_e), 1 input (u) and 1 output (y_e). The cost function in (17) formulated with prediction horizon of, $N = 2$, delta input penalty, $R = 1$, output penalty, $Q = 1$ and the constraints on input as $0 \leq u \leq 24$. The MPC problem is constructed in open source freely available MATLAB® based Multi-Parametric Toolbox (MPT) and an explicit PWA control law (19) was obtained with 13 regions. Further, the sequential search algorithm for the

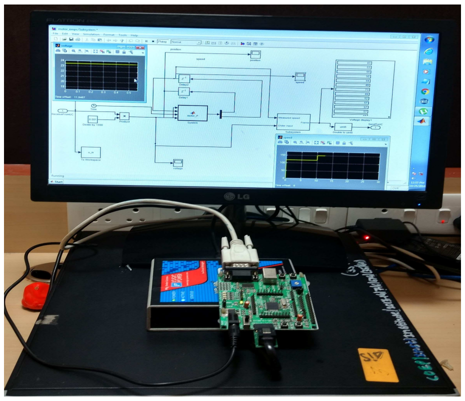

Fig. 2. PIC 18F4550 microcontroller SoC HIL co-simulation lab set-up used in the speed control of DC motor.

controller is exported in the form of a library-free low-level C code. The generated C code is complies with the American National Standards Institute (ANSI) standard and can be used directly in the MPLAB® X IDE. The matrices generated (LUT) using the tool are of data type double and in XC8, the size of double has been configured as $32\,\text{bit}$ in XC8 linker's memory model option.

B. MATLAB®*and Microcontroller Interface*

USART interface module available in PIC 18F4550 has been used for asynchronously communicating with MATLAB/SIMULINK. The data frame as prepared for single byte transfer comprised a start bit and stop bit while $8\,\text{bit}$ of data is sandwiched between them. The baud rate selected for data transfer is $115\,200\,\text{bps}$ and the processor is running at $48\,\text{MHz}$. In the on-line phase, search algorithm needs current value of $\tilde{x}_0$ which is comprised of six variables i.e. position ($\hat{x}_e(t)(1)$), speed ($\hat{x}_e(t)(2)$), estimated speed ($\hat{x}_e(t)(3)$), estimated disturbance ($\hat{x}_e(t)(4)$), previous input ($u_{(t-1)}$) and reference ($r(t)$). All these values are sent to the microcontroller through customized data packet as shown in Fig. 3. For data security reasons, we need to provide the separators (S, I, D, C, O, E and P) for each of the data which has been done by appending characters before and after the variable. So, a frame has been designed in MATLAB® which after being sent will be decoded by the microcontroller and separated into the variables which are required for EMPC evaluation at each sample time. Packet size depends on the accuracy of data required for the application. Data being sent in this application is accurate up to second digit after decimal point. Five bytes of each of the six variables are being sent to the microcontroller.

When frame will be sent, the microcontroller will start accumulating the data in the buffer. After the frame is completely received, then it will begin assigning the variables which would be further sent to EMPC routine for generating optimal input. This input would be later converted into another frame which can be decoded into MATLAB® for obtaining correct input signal. The data being sent to MATLAB® in this application is accurate up to third digit after decimal point. As maximum voltage input to motor is $24\,\text{V}$, $5\,\text{bytes}$

S	$\hat{x}_e(t)(1)$	I	$\hat{x}_e(t)(2)$	D	$\hat{x}_e(t)(3)$	C	$\hat{x}_e(t)(4)$	O	$u_{(t-1)}$	E	$r(t)$	P

Fig. 3. Representation of data frame used to send data from MATLAB/SIMULINK to microcontroller.

of input are being sent. Overall considering the header and terminator byte, and start and stop bits, $70\,\mathrm{bits}$ are being sent.

C. *Realization of Explicit MPC on PIC Microcontroller*

Considered microcontroller PIC 18F4550 does not provide a dynamic memory allocation [18]. Hence, the code was tweaked so as to eliminate the need for dynamic memory allocation which was originally present in code generated by MPT. Following are the C-functions used in the realization of explicit MPC,

```
void HighPriorityInterrupt  IntISR(void).
```

When an interrupt is generated, it would land here in this function (to address 0×08 (high priority interrupt vector address)) from where it would be redirected to `MyBufferRX` function. This is done as the interrupt vector table has limited space.

```
void MyBufferRX(unsigned char).
```

In this function, the received frame is separated and data is converted to an integer from American Standard Code for Information Interchange (ASCII). This array is then sent to `MPCExplicit()` function

```
void MPCExplicit(double *).
```

In this function, the received array is sent to control law evaluation function i.e. `SequentialSerach()`. The input, when received from this function, is multiplied by 1000 and then sent to `Int2AsciiTX(unsigned long int u)`. This is done so that the data, precise up to 3 digits after decimal is obtained.

```
double SeqentialSerach(double *X).
```

This function looks for suitable input value based upon the array of values sent to the function and returns input. This code is generated using MPT toolbox of MATLAB®.

```
void Int2AsciiTX(unsigned long int u).
```

This function converts integer value of the input to ASCII value and transmits data serially to MATLAB®. Then ports initialization and interrupts enabling is done by following function

```
void Initialise().
```

VI. HIL Co-Simulation Results

This section describes the HIL co-simulation results of the implemented explicit MPC and PI controller on the microcontroller. Furthermore, the performance of explicit MPC is shown with the constraint handling, trajectory tracking, disturbance rejection and the complexity of control law is discussed.

A. *Controller Performance*

Fig. 4 shows the output and input response, with explicit MPC and the PI controller, for the varying speed reference tracking. It is clearly seen that the designed MPC achieves reference speed faster compared to PI controller. Input voltage is shown in the below sub-plot, which shows that the input for both the controls are operating within the constraints.

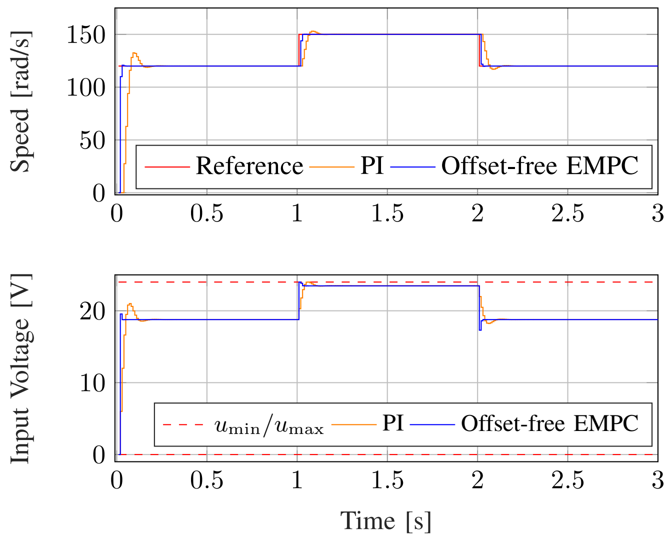

Fig. 4. Performance of PI and offset-free explicit MPC for reference tracking.

Further, offset-free explicit MPC is tested for the disturbance rejection. Fig. 5 shows the response of motor speed control. A disturbance of $10\,\mathrm{V}$ and $20\,\mathrm{V}$ was given at the time $0.5\,\mathrm{s}$ to $0.7\,\mathrm{s}$ and $1.4\,\mathrm{s}$ respectively. Also, to test noisy signal scenario, we added noise up to the amplitude of $8\,\mathrm{V}$ in between time $2.3\,\mathrm{s}$ to $2.7\,\mathrm{s}$. It can be seen that EMPC performs better than PI controller and reject disturbances. The PI controller takes more time to settle and shows oscillations which is not acceptable for the actual motor.

B. *Controller Complexity*

The designed offset-free explicit MPC is implemented on the PIC 18F4550 microcontroller using two different prediction horizons (N) and subsequently, we compare controller complexity on the basis of the number of regions, memory used to store controller data, and memory used for the sequential search and communication interface code. Table II shows the complexity of explicit MPC controller for prediction horizon 2 and 3, for the prediction horizons larger than

TABLE II

COMPARISON OF EXPLICIT MPC CONTROL LAW FOR DIFFERENT PREDICTION HORIZONS.

N	# Regions	Available ROM [kB]	Used ROM [kB]	Usages [%]	Available RAM [kB]	Used RAM [kB]	Usages [%]
2	13	32.77	15.94	48.67	2.05	0.33	16
3	59	32.77	30.01	92.00	2.05	0.33	16

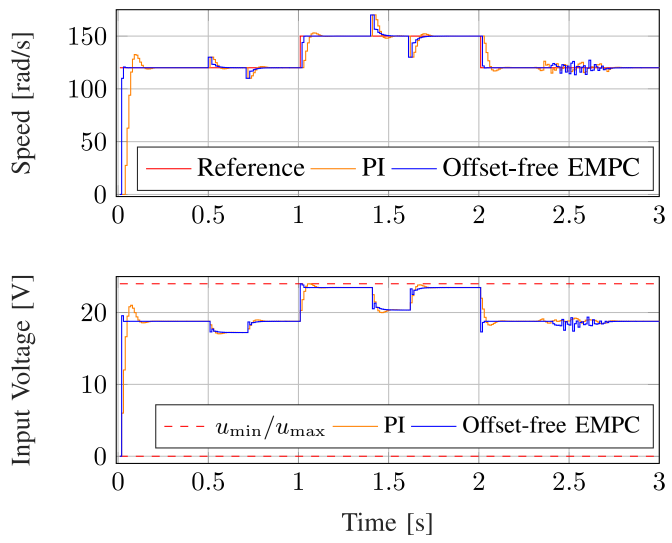

Fig. 5. Performance of PI and offset-free explicit MPC for reference tracking with disturbances.

3 memory exceeds the available on-chip ROM. The explicit MPC matrices (F, g, Z and z) and the code were saved on the ROM and other data need for the implementation was saved on RAM. It can be seen from the table that $N = 2$ needs 43.3% less memory as compared to the $N = 3$ but in the form or performance quality both the controllers give almost same output, that's why explicit controller with $N = 2$ can be the appropriate choice for this application.

VII. CONCLUSIONS

This paper focuses on the implementation of embedded MPC on a low-cost PIC 18F4550 microcontroller. The designed offset-free explicit MPC for the speed control of DC motor with disturbance modeling approach using multi-parametric toolbox which generates the low-level C.It is tweaked and modified so as to run on the microcontroller platform. The results obtained indicate that the EMPC solution outperforms the conventional PI controller with respect to performance and controller efforts to track reference with reasonable utilization of program memory. The HIL co-simulation results obtained from MPT are highly satisfactory and reflects the effectiveness of proposed embedded solution for motor speed control and the potential embedded applications.

ACKNOWLEDGMENT

We gratefully acknowledge the support from R & D center of the COEP. Deepak Ingole would like to thank for a financial contribution from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation program (grant agreement No 646592 MAGnUM project).

REFERENCES

[1] L. G. Bleris and M. V. Kothare, "Real-time implementation of model predictive control," in *Proceedings of the 2005, American Control Conference, 2005.* IEEE, 2005, pp. 4166–4171.

[2] P. D. Vouzis, L. G. Bleris, M. G. Arnold, and M. V. Kothare, "A system-on-a-chip implementation for embedded real-time model predictive control," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1006–1017, 2009.

[3] G. Frison, D. K. M. Kufoalor, L. Imsland, J. Bagterp *et al.*, "Efficient implementation of solvers for linear model predictive control on embedded devices," in *2014 IEEE Conference on Control Applications (CCA).* IEEE, 2014, pp. 1954–1959.

[4] N. Yang, D. Li, J. Zhang, and Y. Xi, "Model predictive controller design and implementation on FPGA with application to motor servo system," *Control Engineering Practice*, vol. 20, no. 11, pp. 1229–1235, 2012.

[5] A. A. Kheriji, F. Bouani, M. Ksouri, and M. B. Ahmed, "A microcontroller implementation of model predictive control," *World Academy of Science, Engineering and Technology*, vol. 5, pp. 5–21, 2011.

[6] A. K. Abbes, F. Bouani, and M. Ksouri, "A microcontroller implementation of constrained model predictive control," *World Academy of Science, Engineering and Technology*, vol. 80, 2011.

[7] T. A. Johansen, "Toward dependable embedded model predictive control," *IEEE Systems Journal". DOI*, vol. 10, 2015.

[8] D. Ingole, J. Holaza, B. Takács, and M. Kvasnica, "FPGA-based explicit model predictive control for closed-loop control of intravenous anesthesia," in *Proceedings of the 20th International Conference on Process Control.* IEEE, 2015, pp. 42–47.

[9] D. Ingole and M. Kvasnica, "FPGA implementation of explicit model predictive control for closed loop control of depth of anesthesia," in *Preprints of the 5th IFAC Conference on Nonlinear Model Predictive Control.* Elsevier, 2015, pp. 484–489.

[10] G. Takács, G. Batista, M. Gulan, and B. Rohal'-Ilkiv, "Embedded explicit model predictive vibration control," *Mechatronics*, vol. 36, pp. 54–62, 2016.

[11] S. Dani, D. Sonawane, D. Ingole, and S. Patil, "Performance evaluation of PID, LQR and MPC for DC motor speed control," in *2017 2nd International Conference for Convergence in Technology (I2CT).* IEEE, 2017, pp. 348–354.

[12] R. Krishnan, *Electric motor drives: modeling, analysis, and control.* Prentice Hall, 2001.

[13] G. Pannocchia and J. B. Rawlings, "Disturbance models for offset-free model-predictive control," *AIChE journal*, vol. 49, no. 2, pp. 426–437, 2003.

[14] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems.* Cambridge University Press, 2017.

[15] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[16] M. Herceg, M. Kvasnica, C. Jones, and M. Morari, "Multi-parametric toolbox 3.0," in *Proceedings of the European control conference*, no. EPFL-CONF-186265, 2013.

[17] M. Kvasnica, J. Holaza, B. Takács, and D. Ingole, "Design and verification of low-complexity explicit MPC controllers in MPT3," in *Control Conference (ECC), 2015 European.* IEEE, 2015, pp. 2595–2600.

[18] T. Microchip, *MPLAB XC8 C Compiler User's Guide*, Microchip Technology Inc., 2015.