Development of a Simulink Dynamic Matrix Control (DMC) Block for Use with an RCP System and Its Application to Motor Control

Young Sam Lee, Jinsuk Choi, Sugkil Seo and Yeong Sang Park
Department of Electrical Engineering, Inha University, Incheon, Korea

Keywords: Dynamic Matrix Control, Rapid Control Prototyping, DC Motor Control, Realtime Control.

Abstract: In this paper, we present the implementation method of a Simulink block for dynamic matrix control (DMC) that can be used in a rapid control prototyping (RCP) environment and consider the speed control of a DC motor using the developed DMC block. Firstly, we introduce a lab-built RCP system briefly. Secondly, we present a method to implement the DMC block using C-language, which enables DMC algorithm to be represented in a library block that can be used in Simulink environment. Finally, we use the developed DMC block for the speed control of a DC motor, through which we show that the DMC-based control system can be easily implemented and it can be applied to real-time control of systems with relatively fast sample rates.

1 INTRODUCTION

Predictive control is the optimization-based control strategy, where we try to find the optimal future control trajectory that minimizes the cost function on the future response predicted using the model information of the controlled system. Predictive control is a computation-intensive control method because it has to solve the constrained optimization problem at each sample time.

In predictive control, we need to construct a predictor to predict the future response of a controlled system. A certain predictive control utilizes the state space model for the predictor (Lee et al., 1994) while the other predictive controls are based on the transfer function model (Clarke et al., 1987a; Clarke et al., 1987b) or on a step response model (Garcia and Morshedi, 1986). Dynamic matrix control, one of many predictive control methods, needs just a step response to construct a predictor while other predictive control requires a state space model or a transfer function model, which is not available in real situations. In that respect, DMC has good practical value.

Merits of the predictive control include the systematic consideration of the input/output constraint of the system and easy extension of its principle to multivariable systems. The ability of handling the input/output constraint comes from the fact that the computation of the control value is formulated into an optimization problem subject to constraints. In most cases, the optimization problems reduce to the quadratic programming (QP) problem. The solution to the quadratic programming is obtained after some complicated procedures and hence the predictive control is computation-intensive methodology. Due to this fact, predictive controls have been majorly used in process control area, where systems are fairly slow in response. Some mathematical software, such as Matlab, provides a command to solve the quadratic programming.

Despite the merits mentioned above, it is not easy for one to apply the predictive control to real control systems. Firstly, one needs the C-code solution to the QP problem for the implementation of the predictive control. qpOASES, which is an open-source QP problem solver, can be good candidate solution to the QP problem (Ferreau et al., 2014). Even with the C-code solution, still the implementation of predictive control in C-code is not an easy task for beginning engineers.

Recently, implementation of the control system is done through RCP systems instead of conventional manual coding. The most representative RCP system includes the one based on Matlab/Simulink. One can construct a controller model using the function blocks supported in Simulink and then translate the controller model into C-codes through the automatic code generation tool. The generated codes can be compiled and downloaded onto a micro-controller board for realtime execution. A certain RCP system does not use the code generation approach. In that system, the Simulink controller model perform as a real-time controller (Lee et al., 2012). One can focus on the de-

Lee, Y., Choi, J., Seo, S. and Park, Y.
Development of a Simulink Dynamic Matrix Control (DMC) Block for Use with an RCP System and Its Application to Motor Control.
DOI: 10.5220/0005958004130420
In *Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2016) - Volume 1*, pages 413-420
ISBN: 978-989-758-198-4

velopment of the control algorithm using the block diagram editing and doesn't have to be worried about the input/output peripheral handling based on manual coding.

In this paper, we propose a method to construct a function block for the dynamic matrix control for use with RCP system based on Simulink. The presented block is implemented in C-code and thus has good computation property. Due to this fact, the resultant control can be applied to systems with relatively fast sample rate. The DMC block can also translated into C-code through the automatic code generation process. In this paper, we will use a lab-built RCP environment for the application of the proposed DMC block.

The paper is structured as follows: In Section 2, we give brief introduction to a lab-built RCP system. In Section 3, we present the method to construct a DMC block. In Section 4, we summarize the procedures about how to construct a speed control system for a DC motor using the DMC block. Finally in Section 5, we make conclusions on the presented results.

2 INTRODUCTION OF THE DAQ RCP

Rapid control prototyping (RCP) system is a kind of development environment that is used for the design, development, and verification of the controller prototype in an efficient way. Generally RCP systems consist of a block diagram based modeling program such as Simulink, library blocks to handle the hardware input/output peripherals, automatic C-code generator, realtime target computer, and a host computer communicating with the target computer. The design and the simulation of the controller are done under Simulink. If the simulation result is satisfactory, the controller is constructed by using the input/output peripheral library blocks. After the code generation of the controller model, the compiled executable is downloaded to a realtime target computer and the realtime control experiment is performed. While the experiment is being performed, one can monitor signals using a host computer connected to a target computer. With the help of the input/output peripheral block library and code generation tool supported in the RCP system, the controller designers have only to focus on the algorithm itself without worrying about the error prone manual coding.

Several RCP systems are available commercially in the market. Matlab/Simulink is the most well known and widely used among those systems. Realtime Workshop (RTW), the add-on product of Simulink, generates C-code for the block diagram-based model constructed by Simulink (Inc., 2005b). Embedded coder, another add-on product of Simulink, generates C-code specific to a certain embedded processor and thus reduce the time for the development (Inc., 2005a). Matlab/Simulink and RTW are open architecture and thus several lab-developed RCP systems for custom-developed hardware have been proposed in academia (Rebeschieß, ; Hong et al., 2000; Lee et al., 2004; Hercog and Jezernik, 2005; Bucher and Balemi, 2006; Lee et al., 2012). Furthermore, several researches related with the application of RCP system are published (Lin et al., 2006; Kennel, 2006).

In this paper, we use the RCP system proposed in (Lee et al., 2012). The RCP system proposed in (Lee et al., 2012) utilizes a DAQ board with high speed USB communication interface and Matlab/Simulink. Because it uses a DAQ board with a microcontroller as a basic component, we will call the system as the DAQ RCP throughout the paper. The DAQ RCP system consists of two subsystems as in Figure 1. The first subsystem is a PC system on which Matlab/Simulink is running and the second subsystem is a DAQ board with high speed USB interface. The PC and the DAQ board communicates the data (control data and sensor data) with each other through the USB communication. Unlike the conventional RCP system where the control algorithm is automatically generated in C-code from the Simulink controller model, the DAQ RCP system has a feature that the Simulink controller model acts as a realtime controller without code generation. Measurement of the sensor data and the application of the control data to output peripherals are taken care of by the DAQ unit with a microcontroller. Figure 2 Figure 2 is the flowchart that shows how the control is performed. Each stage in the flow chart can be summarized as follows:

S1: The DAQ unit measures all the required sensor data and sends them to the PC through USB communication.

S2: Simulink running on the PC receives the data sent from the DAQ unit.

S3: Simulink performs the control computation using the received sensor data.

S4: Simulink sends the computed control data to the DAQ unit.

S5: The DAQ unit applies the received control data from the PC to corresponding output peripherals (DAC, PWM, etc).

It is noted that all the tasks performed in the DAQ unit are handled by the microcontroller. As mentioned in

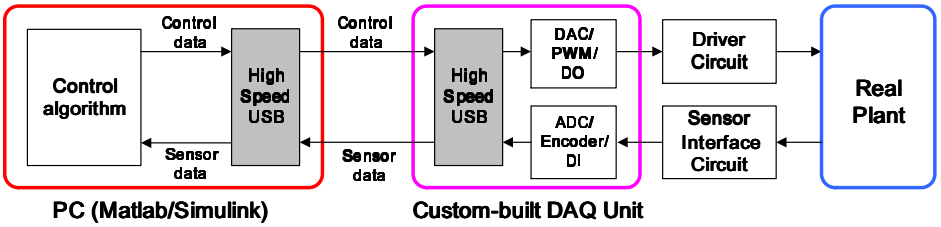

Figure 1: The conceptual diagram on a lab-built DAQ RCP system.

(Lee et al., 2012), the DAQ RCP system can handle the control system with sample rates of up to 2 KHz. Therefore, a good portion of control experiments performed in undergraduate or graduate courses can be supported by the DAQ RCP system. It provides useful input/output function blocks that can handle input/output peripherals in a graphical manner. The representative input/output function blocks include an encoder counter block, a digital input block, an ADC block, a PWM block, a digital output block, and a DAC block.

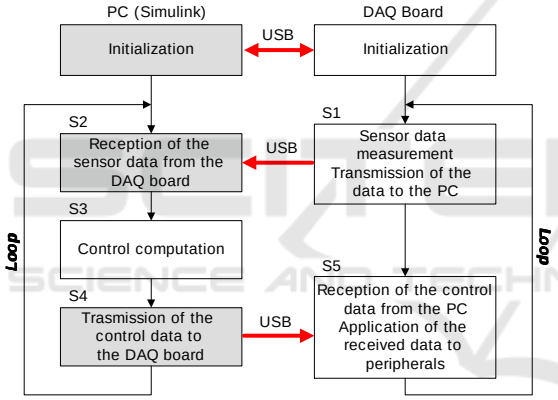

Figure 2: The flow chart of the DAQ RCP system proposed in (Lee et al., 2012).

3 IMPLEMENTATION OF A DMC BLOCK

In this section, we present how to construct the DMC function block using a C-code S-function. The S-function provides a powerful mechanism for extending the capabilities of the Simulink environment. The S-function is a computer language description of a Simulink block written in Matlab, C, C++, or Fortran. One can write an S-Function to describe a user-defined function block. We use a lab-built DAQ RCP system in this paper, where we don't have to generate the C-code for the model. However, to speed up the computation, we develop the DMC block in C-code S-function.

Figure 3 shows the control system diagram that adopts the DMC as a control method. The main purposes of this paper are as follows: to make the DMC be used in Simulink environment by developing the DMC function block in a C-code S-function; to make the DMC be used in the RCP system using the developed DMC block; to give the environment in which one can implement the realtime dynamic matrix control of real systems easily.

If we construct the speed control system for a DC motor using the DMC block in the DAQ RCP system, the controller model will look like Figure 4. To measure the speed of a motor and to generate the PWM signal to a motor driver, we will use the encoder block and the PWM block provided by the DAQ RCP system.

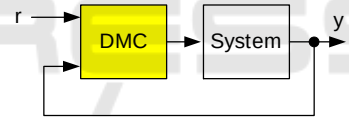

Figure 3: The block diagram of the DMC-based control system.

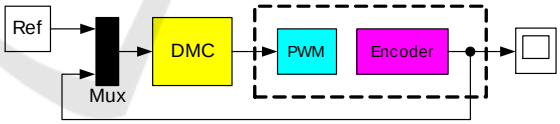

Figure 4: The block diagram of the DC motor control system implemented in Simulink using the DAQ RCP.

The implementation of the DMC block includes three stages. Firstly, we have to implement the solution to the QP problem. Secondly, we have to implement the DMC algorithm using the obtained QP solution. Finally we define the DMC block in a C-code S-Function.

3.1 Implementation of the QP solution

The most essential algorithm to implement the predictive control may be the solution algorithm to the QP problem. The dynamic matrix control is one of predictive control methods and thus we need to have a C-code solution to the QP problem to implement the

DMC block. In this paper, we adopt the method presented in (Lee et al., 2014) to develop the DMC block.

The general QP problem is stated as follows:

$$\min_x \quad \frac{1}{2}x^T Px + q^T x \quad \text{subject to} \quad Ax \le b. \tag{1}$$

With $P > 0$, the problem reduces to a convex optimization and the unique and global solution is always guaranteed. The QP problem to be solved in predictive control satisfies $P > 0$. So we use the solution to the QP problem for the specific case $P > 0$. As explained in (Lee et al., 2014), the QP problem can be equivalently changed into the NNLS(Nonnegative least-squares) problem after a series of equivalent transformations. Therefore one can obtain the solution to the QP problem buy solving the following NNLS problem:

$$\text{NNLS: } \min_d \quad \|Qd - r\| \quad \text{subject to } d \ge 0, \tag{2}$$

where

$$Q = \begin{bmatrix} \bar{A}^T \\ \bar{b}^T \end{bmatrix}, \quad r = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

and $\bar{A} = -AVS^{-1}$, $\bar{b} = -A\bar{q} - b$, $\bar{q} = P^{-1}q$. Matrices V and S are obtained by performing the singular value decomposition (SVD) of $\sqrt{P}$ and they satisfy $\sqrt{P} = USV^T$. Letting the solution to the NNLS problem given in (2) be d^*, the solution to the original QP problem given in (1), x^*, can be given as follows:

$$x^* \quad = \quad VS^{-1}\left[\frac{\bar{A}^T d^*}{(1 - \bar{b}^T d^*)}\right] - \bar{q}$$

The following computations should be done to obtain the solution to the QP problem given the matrices P, q, A, and b:

(a) matrix addition, subtraction, and multiplication

(b) solving the NNLS problem

(c) singular value decomposition of $\sqrt{P}$

(d) matrix inversion to obtain P^{-1}

Computation for (a) can be easily programmed in C code. Solving the NNLS problem in (b) can also be implemented in C code using the algorithm presented in (Lawson and Hanson, 1974). For the computation of (c) and (d), we need to compute the square root of a matrix, singular value decomposition, and matrix inversion, which are somewhat complicated process. However, the matrix P is a constant matrix as long as the system model used to design the predictive control does not change. Therefore, we don't have to implement the C-code to compute (c) and (d). Instead, we use Matlab to compute (c) and (d) and then use the obtained results by generating array data for them in a separate header file. One can use the Matlab command 'fprintf' for that purpose.

3.2 Implementation of the DMC

The DMC, one of predictive controls, utilizes the step response of a system to construct a predictor. The DMC solves the following optimization problem subject to constraints at every sample time to compute the control value:

$$\begin{aligned} \text{Minimize} \quad J = & \sum_{j=1}^{N_p} \|w_{k+j} - \hat{y}_{k+j}\|^2 \\ & + \sum_{j=0}^{N_c-1} \|\Delta u_{k+j}\|_\Lambda^2 \end{aligned} \tag{3}$$

$$\begin{aligned} \text{subject to} \quad & u_{\min} \le u_{k+j} \le u_{\max}, \\ & \Delta u_{\min} \le \Delta u_{k+j} \le \Delta u_{\max}, \\ & j = 0, 1, \cdots, N_c - 1. \end{aligned} \tag{4}$$

The subscripts used in variables denote the time step and k denotes the current time step. Letting u be the control value, u_{k+j} denotes the control value at the moment j step away from the current time. w is the reference input, $\hat{y}$ is the predicted output, Δu is the control increment, N_p is the prediction horizon, and N_c is the control horizon. Generally it holds that $N_p \ge N_c$. Λ, a diagonal matrix, is the weighting on the control increment. Hence the square of the weighted norm can be represented as $\|\Delta u_{k+j}\|_\Lambda^2 = \Delta u_{k+j}^T \Lambda \Delta u_{k+j}$. In the DMC, the control increment is obtained up to the step $k + N_c - 1$ at every sample time. However, only Δu_k is used to obtain the control value, which is given as

$$u_k = u_{k-1} + \Delta u_k.$$

In the DMC, the predicted output from the current step to the moment which is N_p steps away is given by

$$\hat{Y}_k = G\Delta U_k + f_k, \tag{5}$$

where

$$\begin{aligned} \hat{Y}_k &= [\hat{y}_{k+1}^T \quad \hat{y}_{k+2}^T \quad \cdots \quad \hat{y}_{k+N_p}^T]^T \\ \Delta U_k &= [\Delta u_k^T \quad \Delta u_{k+1}^T \quad \cdots \quad \Delta u_{k+N_c-1}^T]^T \end{aligned}$$

and G is a prediction matrix consisting of step response data. f_k is the free response, which can be viewed as the contribution of the past data to the fu-

ture output, is given by

$$G = \begin{bmatrix} g_1 & 0 & \cdots & 0 \\ g_2 & g_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ g_{N_c} & g_{N_c-1} & \cdots & g_1 \\ \vdots & \vdots & \ddots & \vdots \\ g_{N_p} & g_{N_p-1} & \cdots & g_{N_p-N_c+1} \end{bmatrix}, \quad (6)$$

$$f_k = \begin{bmatrix} \bar{y}_k \\ \bar{y}_k \\ \vdots \\ \bar{y}_k \end{bmatrix} + M \begin{bmatrix} \Delta u_{k-1} \\ \Delta u_{k-2} \\ \vdots \\ \Delta u_{k-(N-1)} \end{bmatrix} + \begin{bmatrix} h \\ 2h \\ \vdots \\ N_p h \end{bmatrix} u_{k-N}, \quad (7)$$

where

$$M = \begin{bmatrix} g_2 - g_1 & g_3 - g_2 & \cdots & g_N - g_{N-1} \\ g_3 - g_1 & g_4 - g_2 & \cdots & g_{N+1} - g_{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{N_p+1} - g_1 & g_{N_p+2} - g_2 & \cdots & g_{N_p+N-1} - g_{N-1} \end{bmatrix}$$

and $\bar{y}_k$ is the measurement of the output, g_i is the step response value at time step $k = i$ obtained by applying a step input to the system at time $k = 0$. In case of multivariable system with n_u inputs and n_y outputs, g_i is a $n_y \times n_u$ matrix (Lundstrom et al., 1995). N is the number of steps taken to reach the steady state when the system is stable or it includes an integrating process. In case of systems with integrating process, the output at the steady state will show a constant slope with time. At steady state, $h = g_{i+1} - g_i$. It holds that $h = 0$ for stable systems and and $h \neq 0$ for systems with integrating process. G does not have any subscript because it is constant. Let's define the future reference trajectory as follows:

$$W_k = [w_{k+1}^T \quad w_{k+2}^T \quad \cdots \quad w_{k+N_p}^T]^T.$$

Then, as explained in (Lee et al., 2014), obtaining DMC control value reduces to a QP problem with ΔU_k being an optimization variable. Here, the matrices P, q, A, and b are given as follows:

$$P = G^T G + \Xi, \quad q = -(W_k - f_k)^T G,$$

$$A = \begin{bmatrix} I_D \\ -I_D \\ I_L \\ -I_L \end{bmatrix}, b = \begin{bmatrix} I_V u_{\max} \\ -I_V u_{\min} \\ I_V (u_{\max} - u_{k-1}) \\ -I_V (u_{\min} - u_{k-1}) \end{bmatrix} \quad (8)$$

where

$$\Xi = \mathrm{diag}\{\Lambda, \Lambda, \cdots, \Lambda\},\ I_D = \begin{bmatrix} I_m & 0 & \cdots & 0 \\ 0 & I_m & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I_m \end{bmatrix},$$

$$I_L = \begin{bmatrix} I_m & 0 & \cdots & 0 \\ I_m & I_m & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ I_m & I_m & \cdots & I_m \end{bmatrix},\ I_V = \begin{bmatrix} I_m \\ I_m \\ \vdots \\ I_m \end{bmatrix}$$

and these matrices have appropriate dimensions depending on the prediction horizon and the control horizon. As shown in (8), among the matrices P, q, A, b pertaining to the QP problem to obtain the DMC control value, P and A remain unchanged while q and b change at every step. Therefore, we can obtain the data related with P and A offline using Matlab and then generate a C-code header file containing the data.

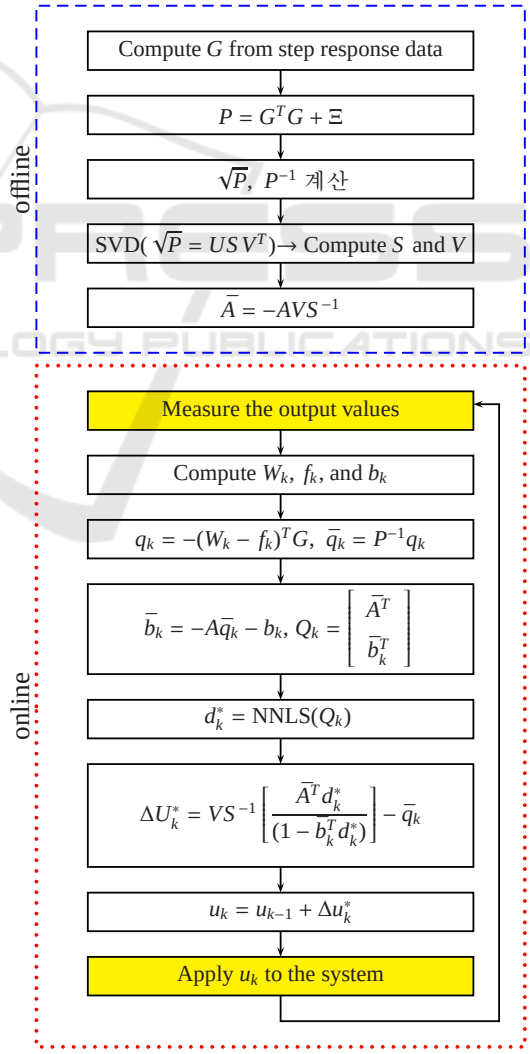

Figure 5: The flow chart of the algorithm.

Furthermore, the generation of q and b should be done online. To explicitly denote that q and b vary at each time, let's denote them by q_k and b_k, respectively. Figure 5 shows the flowcharts for the offline computation part and the online computation part. We can implement the DMC algorithm by translating the algorithm in the flowchart in C-code.

3.3 Implementation of the DMC Block

We can implement the DMC algorithm presented above in C-code. However, it is never easy for students or engineers who are not familiar with control theory to implement the DMC algorithm in C-code by themselves. If we can provide a function block for DMC that can be used in a Simulink environment, one has only to copy the DMC block to build a controller in a graphical manner. In this respect, we implement the DMC function block that can be used in Simulink. Matlab allows the users to define a custom function block through the S-function. In an S-function, one has to define a block initialization function, a output function, a termination function, etc to describe the every aspect of a block. As in the concept diagram given in Figure 4, the inputs to the DMC block are the reference input and the measured output and the output of the DMC block is the control value. The relationship between the inputs and output was described in the previous subsection.

4 IMPLEMENTATION OF THE DMC-BASED VELOCITY CONTROL FOR A DC MOTOR

In this section, we will deal with the realtime dynamic matrix control of the speed of a DC motor by constructing a controller using the DMC block in the lab-built RCP environment. By doing so, we will illustrate that the DMC-based control system can be easily constructed and can be applied to systems with somewhat fast sample rates. We use a 150W Maxon DC motor in the experiment. It has a rotary incremental encoder with the resolution of 500 PPR to measure the position and speed of a rotor. Figure 6 shows the experimental setup, where the left part is a DAQ unit with an Arduino Due being a main microcontroller and the center is a DC motor driver using an H bridge, and the right part is a Maxon DC motor. The DAQ unit is connected to the PC through the USB interface. We use the unipolar complementary PWM to drive the motor driver.

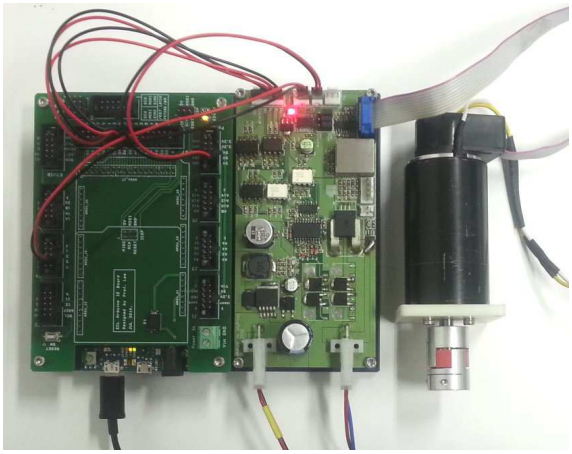

Figure 6: The experimental set for the speed control of a DC motor.

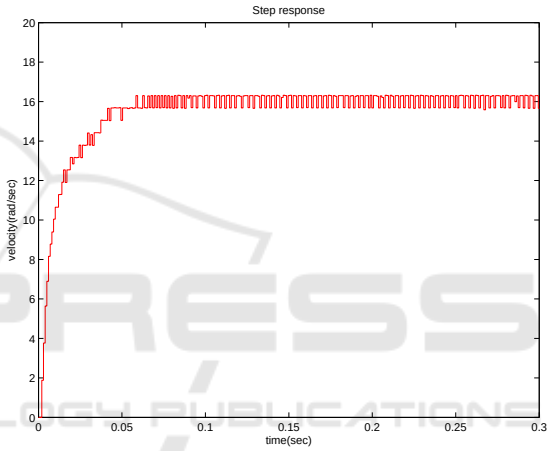

Figure 7: A step response (raw data).

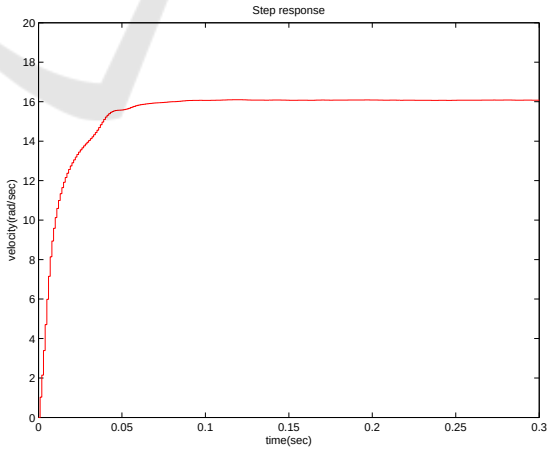

Figure 8: A step response after post processing.

4.1 Acquisition of the Step Response Data

In order to construct a DMC-based controller, we first have to obtain the step response data of a system. For

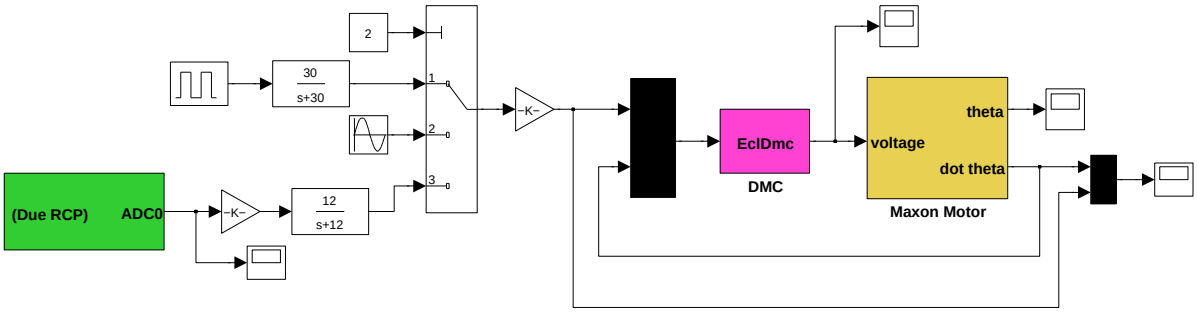

Figure 9: DC motor speed control system implemented using a developed DMC block.

this purpose, we applied a step input of 5 volt to the motor and measured the resultant speed. We obtained a unit step response through the normalization (i.e., division by 5). The sample rate of the Simulink model is set to 1 KHz, which is relatively fast for DMC-based control system.

It is not easy to get the reliable step response data through just one experiment. Figure 7 shows the step response data that has not gone through any post processing. The rotary encoder has limited resolution and thus quantization noise is clearly seen in the figure. We performed 30 experiments and we compute the average. Furthermore, we post-processed the data using a noncausal filter, which is supported by the Matlab command 'filtfilt', to get smoother step response data. Figure 8 shows the step response data that has gone through this post-processing.

4.2 DMC-based Speed Control

We can decide the horizon N in (7) from the moment when the step response reaches the steady state. From Figure 8, it is seen that the step response goes into the steady state from $t = 0.3$. Because the sample time is 1 ms, we chose $N = 0.3/0.001 = 300$. The control horizon and the prediction horizon were chosen to be $N_c = 3$, $N_p = 3$. We set the DC power supply so that its output is 10 V. This leads to $u_{\max} = 10$ and $u_{\min} = -10$. Figure 9 shows the Simulink speed control model implemented using the proposed DMC bloc. It is constructed so that we can choose one out of three possible reference trajectories, which are a square wave, a sinusoidal wave, and external input coming from the potentiometer. In Figure 9, a sinusoidal wave is chosen as the speed reference input.

Figure 10 shows the experimental result of the DMC-based speed control with the sinusoidal wave being a reference input. The blue line is the reference input, the red line is the actual speed of the motor. It is clearly seen that the motor speed tracks the speed reference successfully.

The sample rate of this experiment is 1 KHz. Considering that the predictive control has been majorly applied to sluggish systems, this sample rate can be said fairly fast. This fast sample rate can be made handled because the DMC block presented in this paper is implemented through a C-cod S-function. Furthermore, we see that the DMC method can be easily implemented by using the presented DMC block in the RCP environment.

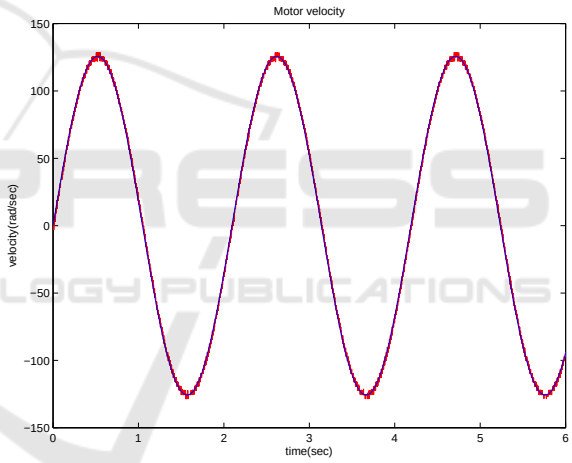

Figure 10: Speed control response of DMC.

5 CONCLUSIONS

In this paper, we developed a Simulink function block for the dynamic matrix control and applied the developed block to the speed control of a DC motor. For the development of a DMC block, we firstly presented how to program a solution to the quadratic programming in C language. We also presented how to implement the DMC algorithm using the presented solution. Furthermore, we implemented a DMC block that can be used in Simulink environment using a C-code S-function. By developing a DMC function block for Simulink and using it in RCP system, the a realtime dynamic matrix control of a system can be easily implemented. In order to illustrate the usefulness of the

proposed DMC block, we applied it to the speed control problem of a DC motor. Through the experiment, we showed that the proposed DMC block can be applied to the realtime control system with relatively fast sample rate.

ACKNOWLEDGEMENTS

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2016-H8601-16-1003) supervised by the IITP(Institute for Information & communications Technology Promotion)

REFERENCES

Bucher, R. and Balemi, S. (2006). Rapid controller prototyping with Matlab/Simulink and Linux. *Control Engineering Practice*, 14:185–192.

Clarke, D. W., Mohtadi, C., and Tuffs, P. S. (1987a). Generalized predictive control-Part I. The basic algorithm. *Automatica*, 23(2):137–148.

Clarke, D. W., Mohtadi, C., and Tuffs, P. S. (1987b). Generalized predictive control-Part II. Extensions and interpretations. *Automatica*, 23(2):149–160.

Ferreau, H. J., Kirches, C., Potschka, A., Bock, H. G., and Diehl, M. (2014). qpOASES: a parametric active-set algorithm for quadratic programming. *Math. Prog. Comp.*, 6:327–363.

Garcia, C. E. and Morshedi, A. M. (1986). Quadratic programming solution of dynamic matrix control (QDMC). *Chem. Eng. Commun.*, 46:73–87.

Hercog, D. and Jezernik, K. (2005). Rapid control prototyping using Matlab/Simulink and a DSP-based motor controller. *Int. J. Engng Ed.*, 21(3):1–9.

Hong, K. H., Gan, W. S., Chong, Y. K., Chew, K. K., Lee, C. M., and Koh, T. Y. (2000). An integrated environment for rapid prototyping of DSP algorithms using and Texas Instruments' TMS320C30 . *Microprocessors and Microsystems*, 24(7):349–363.

Inc., T. M. (2005a). *Real-time workshop embedded code user's guide (ecoder_ug.pdf), Version 4*.

Inc., T. M. (2005b). *Real-time workshop user's guide (rtw_ug.pdf), Version 6*.

Kennel, R. (2006). Improved direct torque control for induction motor drives with rapid prototyping system. *Energy Conversion and Management*, 47:1999–2010.

Lawson, C. L. and Hanson, R. J. (1974). *Solving least squares problems*. Prentice-Hall, Englewood Cliffs, New Jersey.

Lee, J. H., Morari, M., and Garcia, C. E. (1994). State-space interpretation of model predictive control. *Automatica*, 30(4):707–717.

Lee, W., Shin, M., and Sunwoo, M. (2004). Target-identical rapid control prototyping platform for model-based engine control. *Proc. Instn Mech. Engrs Part D, J. Automobile Engineering*, 218:755–765.

Lee, Y. S., Gyeong, G. Y., and Park, J. H. (2014). QP Solution for the implementation of the predictive control on microcontroller systems and its application method. *Journal of Institute of Control, Robotics, and Systems (in Korean)*, 20(9):908–913.

Lee, Y. S., Yang, J. H., Kim, S. Y., Kim, W. S., and Kwon, O. K. (2012). Development of a rapid control prototpying system based on Matlab and USB DAQ boards. *Journal of Institute of Control, Robotics, and Systems (in Korean)*, 18(10):912–920.

Lin, C. F., Tseng, C. Y., and Tseng, T. W. (2006). A hardware-in-the-loop dynamics simulator for motorcycle rapid controller prototyping. *Control Engineerning Practice*, 14:1467–1476.

Lundstrom, P., Lee, J. H., Morari, M., and Skogestad, S. (1995). Limitations of dynamic matrix control. *Computers Chemical Engineering*, 19(4):409–421.

Rebeschieß, S. MIRCOS-Microcontroller-based real time control system toolbox for use with Matlab/Simulink. In *Proc. IEEE Int. Symp. Computer Aided Control System Design*, pages 267–272, 1999.