Model-based predictive controller design

Soheil Sadeghi
University of Northern Iowa

Model-based predictive controller design

Abstract

Process control in industries is becoming more critical due to demands on reducing consumed energy, reducing cost, and improving system efficiency and performance. In this work, a well-developed, model-based controller is introduced and implemented in one of the most common processes in the industry. Model Predictive Controller (MPC) has been studied for ten years, but recently has emerged into small industries as the number of current applications grows. These controllers need a process model in the forms of transfer functions, step response, or state-space. Although obtaining a process model can be a cumbersome task, it is worth having it (the model includes a lot of information about the process). Regarding available resources to build a test station, complexity of the test procedure, involvement in system identification, system study, and comparison to other control algorithms, this work is the first step that must be taken. The first step is limited to a linear single input—single output system. The MPC algorithm is exploited and introduced using coding in two different environments: MATLAB and an embedded system using C code. Results of simulation and real lab measurement will be compared, and limitations and future work are also addressed.

Model-Based Predictive Controller Design

A Graduate Research/Project Paper
Presented to the Graduate Faculty of the Department of Technology
University of Northern Iowa

In Partial Fulfillment of the Requirements for the
Non-Thesis Master of Science in Technology Degree

By
Soheil Sadeghi
November 2021

Approved by:

Dr. Jin Zhu______________________________
Signature of Advisor Date

Dr. Ali Kashef______________________________ 11/19/2021
Signature of Second Review Date

Abstract

Process control in industries is becoming more critical due to demands on reducing consumed energy, reducing cost, and improving system efficiency and performance. In this work, a well-developed, model-based controller is introduced and implemented in one of the most common processes in the industry. Model Predictive Controller (MPC) has been studied for ten years, but recently has emerged into small industries as the number of current applications grows. These controllers need a process model in the forms of transfer functions, step response, or state-space. Although obtaining a process model can be a cumbersome task, it is worth having it (the model includes a lot of information about the process). Regarding available resources to build a test station, complexity of the test procedure, involvement in system identification, system study, and comparison to other control algorithms, this work is the first step that must be taken. The first step is limited to a linear single input−single output system. The MPC algorithm is exploited and introduced using coding in two different environments: MATLAB and an embedded system using C code. Results of simulation and real lab measurement will be compared, and limitations and future work are also addressed.

Table of content

Introduction

Controllers are an inseparable part of process control industries. They are used to track the input, eliminate or alleviate measurement noise, and reject disturbances. In medium- to large-sized industries, every amount of thermal energy can be converted to mechanical work, one way is to increase entropy by increasing controllers' performance. Saving energy can help preserve the world's resources. Large industries like power plants, oil and petrochemical refineries, pharmacological plants, and car and semiconductor companies increase their controller performance to achieve a higher performance and preserve existing resources. Controller performance is key to the reduction of final product cost and helps save companies money.

Controllers can be categorized as non-model-based controllers and model-based controllers. The non-model-based controllers are not inherently capable of minimizing a cost function. In contrast, model-based controllers are kept growing since they can naturally reduce cost functions (Khaled & Pattel, 2018), This cost function could be energy used by the process. Another advantage of model-based controllers is handling multi-inputs/outputs processes like the car engine oil industry, which may have around 50 inputs-outputs (Seborg, Edgar, & Mellichamp, 2004). One of the popular industry controllers is PID (proportional integrating derivative). Ease of use, cost-effectiveness, simple structure, fast response, and the fact that it is well-studied (Åström, Hägglund, 1995). PIDs are non-model-based controllers that don't need a process transfer function. One of the model-based controllers is Model Predictive Controllers (MPC) (Bemporad, 2006). This algorithm has been around for many years, but has gathered new attention due to two factors:

1. Rapid-growing applications
2. Availability of high-speed microcontrollers

MPC is a numerical algorithm that takes care of processes with:

- Multiple inputs/outputs (MIMO) systems
- Input to a few outputs' interactions
- Constraints on input/output values and rate of change
- Long time delays
- Nonlinearity
- Cost and object function minimization

Most of the time, literature takes MPC and PID controllers and compares them side-by-side (Jibril, 2020). MPC is from the category of the model-based controller and PID from the non-model-based controller. Since PIDs don't need to know any information about the process, they cannot take full advantage of the controller’s capability. This is one of the main disadvantages of PID controllers. It is observed that the output of PID becomes saturated - meaning, it goes to its maximum or minimum for a particular time. At this moment, closed-loop feedback becomes an open loop, and there will be no control action on the process. In MPC, we can define input/output and min/max values to avoid this problem. To differentiate between the MPC application and PID, let's look at the two figures below. Figure 1 shows a PID usually utilized in a SISO (Single Input / Single Output) process. Figure 2 shows a process adopted for MPC in the pharmaceutical industry, which is a typical application of MPC.

Figure 1

Single input single output simple process with PID controller (Sivaram et al., 2021)

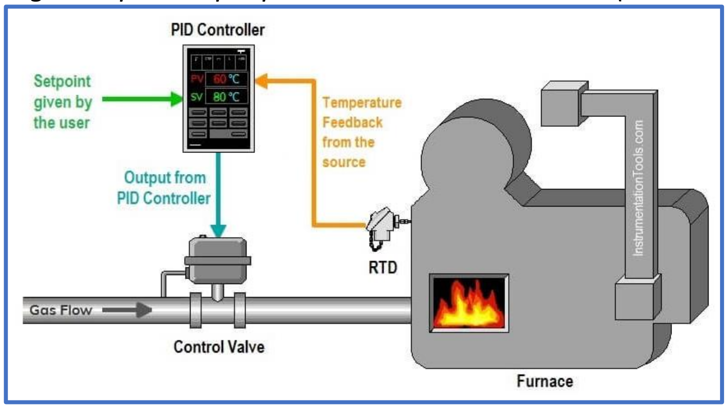
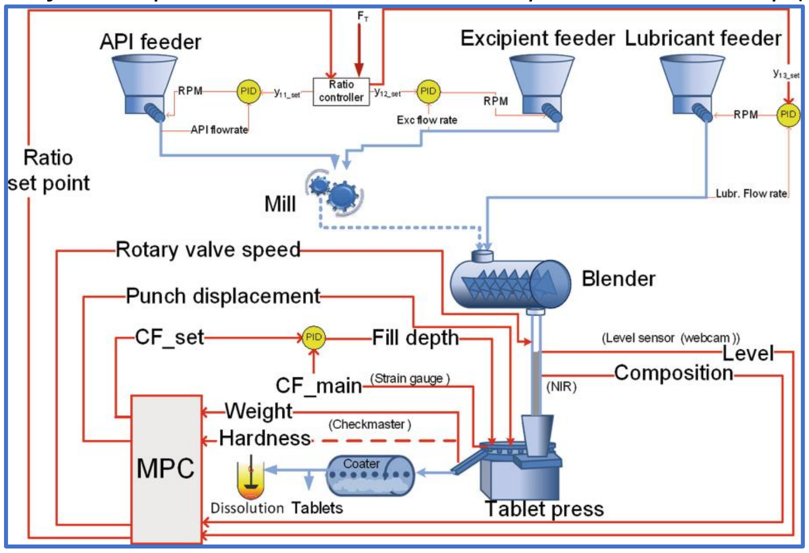

PID Controller
Setpoint given by the user
PV 60 °C
SV 80 °C
Temperature Feedback from the source
Output from PID Controller
RTD
Gas Flow
Control Valve
Furnace
InstrumentationTools.com

Figure 2

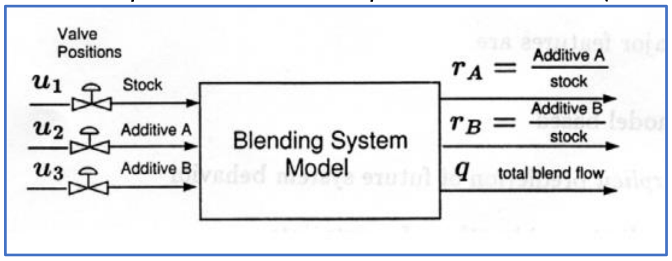
Four inputs and four outputs with MPC structure in a process control loop (Singh, 2018)



As you see in Figure 2, input-output to controller block (MPC) can be quite complicated, this includes coupling between one input to a few outputs, and a different number of inputs to outputs (e.g., 4 inputs and 3 outputs); while in PID, the number of inputs and outputs are the same. Another reason for PID's shortcomings is gain-tuning. In PID, three parameters must be determined by the control engineer or operator to match the design specifications, like an overshoot of five percent, a settling time of two minutes, and a steady-state error of one percent. The tuning method is not a simple task if you consider a wide range of setpoints, while each setpoint needs a set of different gains **(*K, T_d, T_i*)**. Many numerical methods, from manual to auto-tuning, are available from PID handbooks to tune this traditional controller. PID in multi- and nested-loop processes has limited applications. We can use them in two-by-two processes without coupling, or in interactions of inputs to output (Khaled & Pattel, 2018). More applications of MPC include stir tanks in chemical reactors that have several input-outputs, signal values that can exceed minimum/maximum threshold values, and interaction that occur between inputs and outputs. A stir tank that has three inputs and three outputs is depicted in Figure 3 below.

Figure 3

MIMO system with three inputs and three outputs in a stir tank (Chikkula et al., 1995).



Purpose of study

In this work, a linear time-invariant MPC is studied for the first-order SISO process. The goal is to build an MPC in an actual experiment to differentiate its advantages over traditional PID controllers and take the first necessary step into more advanced model-based controllers. The algorithm presented in this work can be extended to higher-order systems as well. To entirely take advantage of this type of MPC controller, certain steps must be taken because some benefits only emerge in particular steps. These steps are:

1. Implementing an MPC controller for single input – single output system

2. Realizing an MPC controller for two inputs – two outputs system
3. Creating a three or more inputs – three or more output system and test MPC algorithm
4. Finally, employing a system that has known or unknown interactions between different inputs to outputs.
5. Engaging in a nonlinear system with interaction is the ultimate test that needs to be performed.

This work is the first step of a series of steps needed to arrive at a complete MPC algorithm and implementation. Step one is applying MPC to a linear first-order system process. The next step is performing MPC to a multiple inputs-output system. In the end, we need to modify the algorithm and test it for nonlinear systems. For each further step, source C code or MATLAB statements have to be changed and tested. One of the incredible results and immediate outcomes of accomplishing step 1 (this work) is the simple tuning of the controller. As we will see, the MPC controller needs to adjust two parameters instead of three in PID controllers. Moreover, this step shows MPC tuning is very smooth and has low sensitivity to controller parameter changes. Since resources are limited, the work done in this project includes step 1 only. Further steps need more resources and time, and could be an opportunity for future work.

Research Questions

The following research questions will be answered in this paper:

1. How different would the process output be between the PID process control and MPC based process control?
2. How comparable are the simulation results from a MATLAB implemented MPC controller and the actual experimental results from an MPC controller implemented using C in an embedded system?

Literature Review

Some literature discussed the trends, MPC tools, and future of this controller. Bemporad (2006) reviewed the basic ideas of MPC design, from the traditional linear MPC setup based on quadratic programming to more advanced explicit and hybrid MPC, It also highlighted available software tools for the design, evaluation, code generation, and deployment of MPC controllers in real-time hardware platforms. In industrial process control, the Honeywell industrial MPC controller was designed to handle complex industrial process control that cannot be dealt with using Practical Design or traditional and popular PID (Khaled & Pattel, 2018). Linear Model Predictive Control (MPC) continues to be the technology of choice for constrained, multivariable control applications in the process industry. Successful deployment of MPC requires "getting right" multiple aspects of the control problem (Darby & Nikolaou, 2012).

Researchers focus on developing and applying Lyapunov-based economic model predictive control (LEMPC) designs to a catalytic alkylation of the benzene process network, which consists of four continuously stirred tank reactors and a flash separator (Chen et al., 2012). This paper (Forgione et al., 2020) tackles the embedded MPC design problem using a global, data-driven optimization approach to consider closed-loop performance and real-time requirements. Showcase this approach's potential by tuning an MPC controller on two hardware platforms characterized by vastly different computational capabilities. Simulation studies on a realistic example show that it is possible to implement constrained MPC on an FPGA chip with a 25MHz clock and achieve MPC implementation rates comparable to those achievable on a Pentium 3.0 GHz PC (Ling et al., 2008). Several articles and textbooks introduce MPC from the primary level to advanced topics, and they are listed below in Table 1.

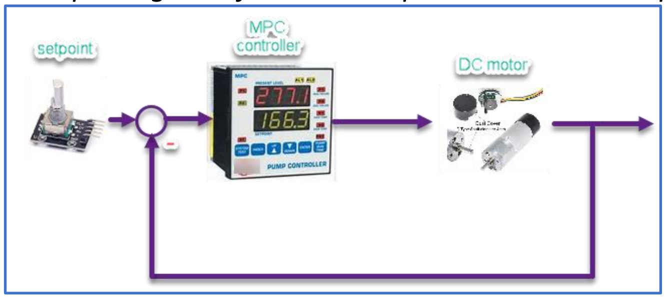
Table 1
Fair sources for predictive controller

1	**Book**	Practical design and application of Model Predictive Control MPC for MATLAB® and Simulink® Users	Nassim Khaled, Bibin Patel, 2018
2	**Book**	Microcontroller based applied digital control	Dogan Ibrahim, 2007
3	**Textbook**	Process dynamics and control, chapter 20	Seborg, Edgar, Mellichamp, Doyle 4th edition, 2017
4	**Textbook**	Systems and Control	Stanislaw H. Zak
5	**Course website**	Purdue university – ECE 680	Stanislaw H. Zak

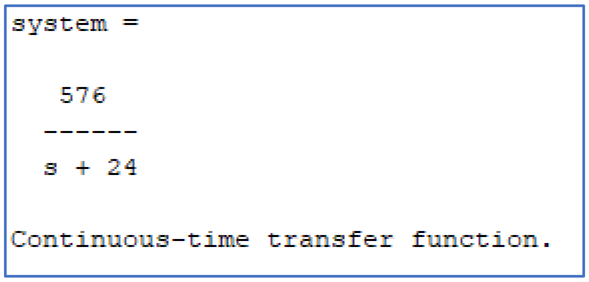
Method of Study

This project examined an SISO system with a simple system order one or two platforms, like a DC motor speed (first-order). A speed controller (RPM-Round Per Minute measurement) is chosen because it is widespread in industries. It has its challenges, such as a fast sampling interval. Its response requires minimal settling time and overshoot (in contrast with another common practice that is boiler temperature control with a considerable time constant and a low sampling interval of as little as five minutes). System identification is the prior knowledge needed to move further in MPC controller design. In this work, MATLAB identification packages have been used to identify 300RPM, DC motor at 12v, and the encoder motor transfer function (ljung, 2012).

Figure 4
single-simple negative feedback loop to control motor speed



In Figure 4, MPC is realized by an ARM microcontroller and C code uploaded into its flash memory. A 0-1v DC motor at a variable speed of 0-300 RPM, and a device to adjust the setpoint, (a rotary encoder switch) are also used. The C programming environment is STM32CubeIDE from ST-microelectronics, and the MCU is STM32L476 Nucleo board at 80 MHz with 1MB RAM. The DC motor has been mathematically identified as a first-order system with its transfer function shown in Equation 1 below.

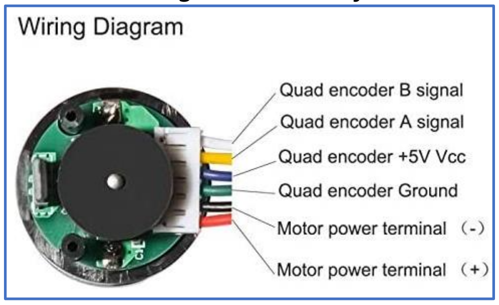


```
system =

   576
  ------
  s + 24

Continuous-time transfer function.
```

(equation 1)

In the above transfer function, input is DC voltage and output is RPM. At 12.5v input, the maximum RPM is 300. A 140-slots hall effect sensor is attached to the motor from the time of purchase to measure motor speed. Please refer to Figure 5 for the quadrature encoder and DC motor wiring diagram. A and B signals determine clockwise or counterclockwise motor rotation direction.

Figure 5
Encoder wiring schematic of DC motor



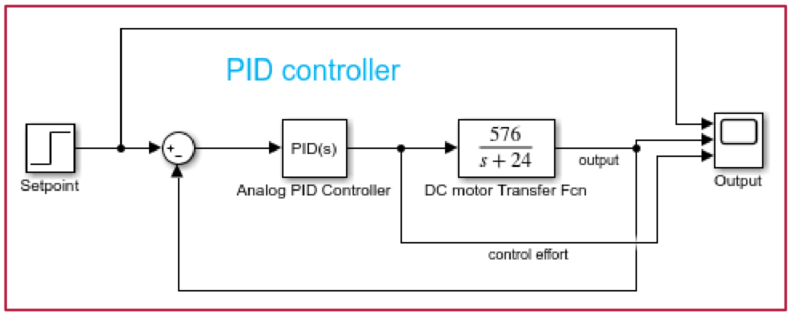
In this work, the following steps were conducted:

1. **Simulink design method and standpoint.**

Simulink step input analysis of the control loop is depicted in Figure 6 for the PID controller, and Simulink MPC unit step input analysis is graphed in Figure 7.

Figure 6

PID DC motor analysis and its output



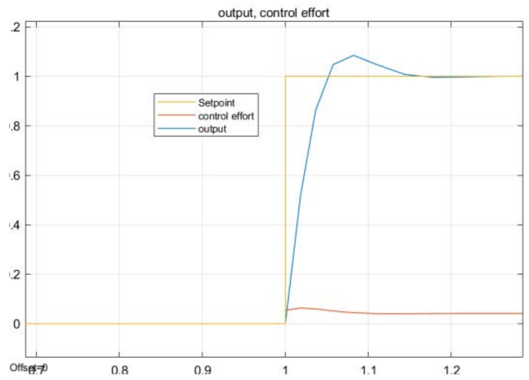
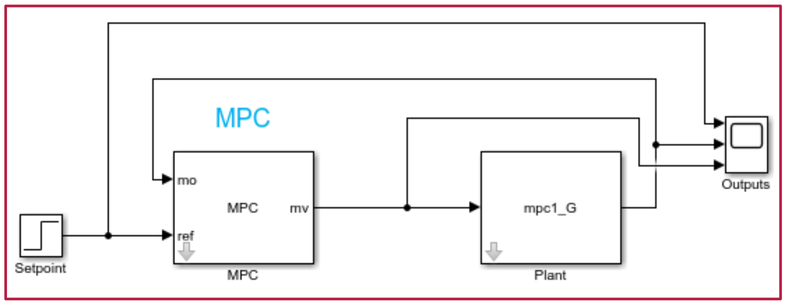
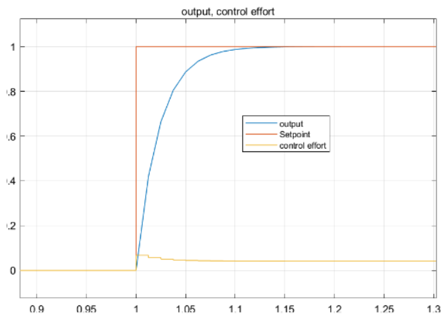


Figure 7

MPC analysis of the DC motor speed loop and results

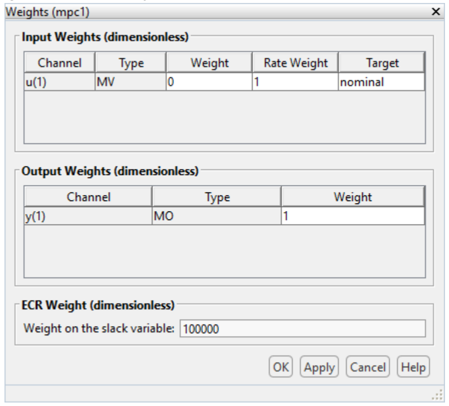




There are no significant differences between the two controllers' output results. We can obtain the same results by tweaking gains in the PID controller and tweaking prediction and control horizons in MPC. The small amount of overshoot in PID response can be removed by gain adjustment. For SISO systems for both controllers, we can define and minimize a cost function. For MPC cost function is defined in the following Simulink window (Figure 8):

Figure 8

Built-in cost function optimization window in Simulink



Weights (mpc1)

Input Weights (dimensionless)

Channel	Type	Weight	Rate Weight	Target
u(1)	MV	0	1	nominal

Output Weights (dimensionless)

Channel	Type	Weight
y(1)	MO	1

ECR Weight (dimensionless)

Weight on the slack variable: 100000

To minimize the cost function for PID, we need to write some MATLAB offline code and use the *fminsearch* command. This cannot be done in real-time by embedded C code for two reasons: the algorithm may not converge, and we have to change the search domain manually otherwise the convergence time to perform search may exceed limits. For these reasons, we usually say that PIDs cannot perform object function minimization, but MPC can.

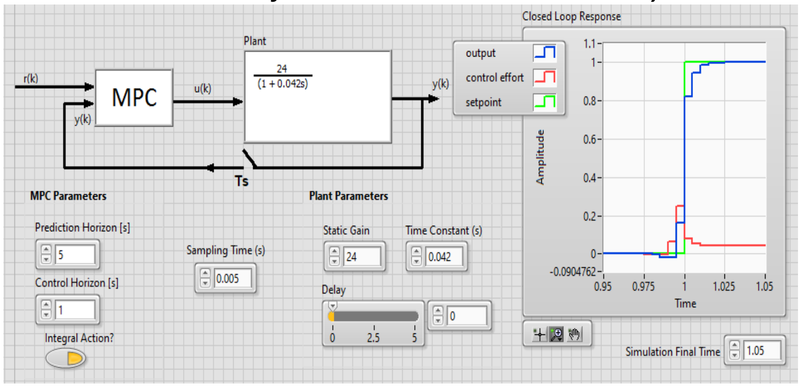
2. LabVIEW MPC

MPC has been integrated into LabVIEW extensively. Before getting deep into predictive control mathematics in the next section, let's look at how to use DAQmx functions to access the Data Acquisition board (DAQ device) and run a LabVIEW VI file. Figure 9 shows the front panel screen and simulation outputs. We incorporated our DC motor transfer function (Equation 1) into the example file of Figure 9. MPC block is an internal LabVIEW function, and other parameters include a 5-millisecond sampling time and a zero time delay, Prediction and control horizons are 5 and 1 respectively. We can realize there are no significant differences between LabVIEW results (blue curve in Figure 9) and Simulink simulation outputs (blue curve of Figure

7). As a result, we will continue with the MATLAB package since we can write our MPC code and compare the result with this package's output.

Figure 9

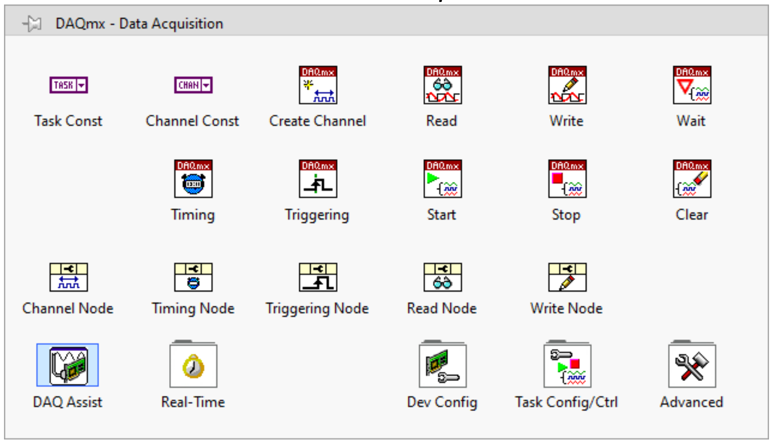
CDEx MPC Transfer Function Model with Delay.vi



To implement embedded hardware-in-loop (HIL) MPC and PID in LabVIEW, and run a DC motor experiment in an online fashion like the one in Figure 4, a DAQ device like USB-6001 is connected and used in the following measurement I/O functions (Figure 10) in LabVIEW. To employ each PID or MPC, please find it under the Control & Simulation function.

Figure 10

I/O function to run a real-time motor speed controller in LabView



3. MATLAB code for MPC realization

Both MPC and PID function calls employed in the above two simulator packages are built-in functions. The source code is not revealed nor available. Therefore, the MATLAB code for the MPC function needs to be prepared before converting it to the predictive controller in C code. A summary of the steps is depicted in Table 2 for clarification:

Table 2

Study steps from MPC mathematics formalization to the C code programing

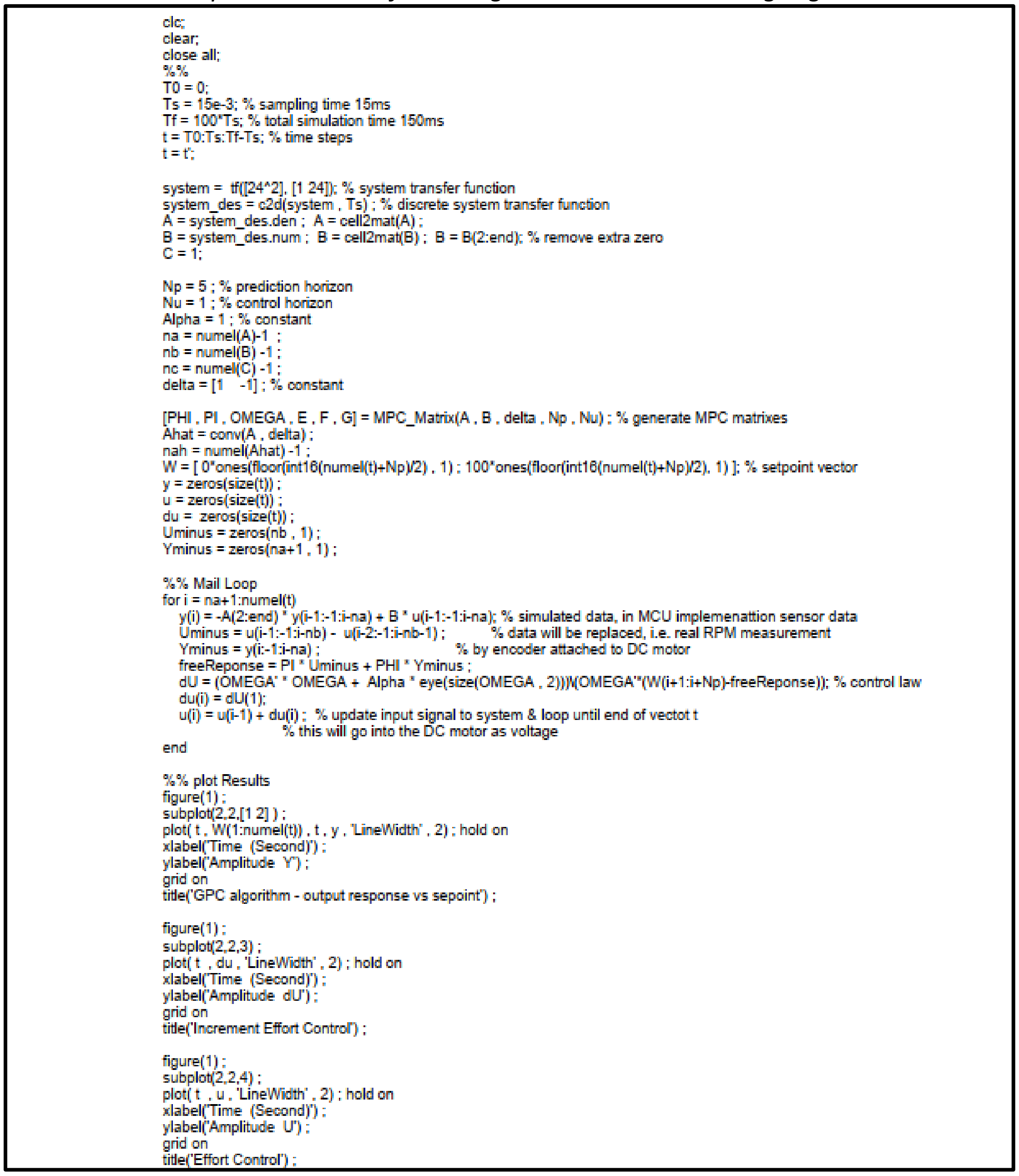
Predictive mathematics	→	MATLAB code	→	Test & verify	→	Embedded C code

Figure 11 demonstrates the source code for MPC in MATLAB language. There are three major categories in the code. The first is Matrix augmentation, the second is cost function optimization, and the third is control effort value update. More detail is listed in the following statements:

- Introduce analog process transfer function of DC motor
- Make the discrete process transfer function
- Make state-space matrices (A, B, C, D)
- Select a prediction and control horizon constants
- Create few intermediate variables, including a constant array
- Create augmented MPC matrixes
- Do incremental update control
- Set the signal input to the system
- Display the results

Figure 11

Implementation of MPC algorithm in MATLAB language

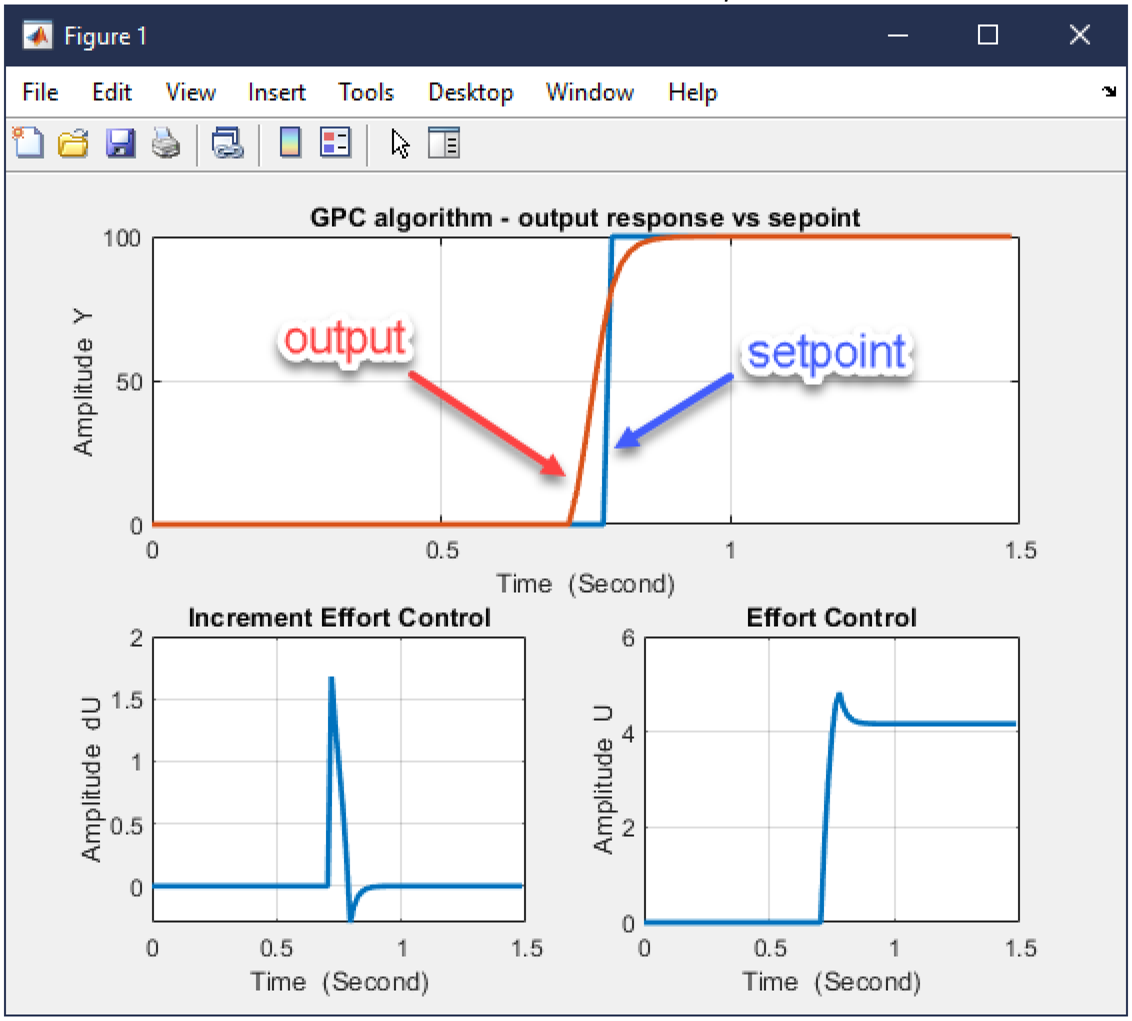


```
clc;
clear;
close all;
%%
T0 = 0;
Ts = 15e-3; % sampling time 15ms
Tf = 100*Ts; % total simulation time 150ms
t = T0:Ts:Tf-Ts; % time steps
t = t';

system =  tf([24^2], [1 24]); % system transfer function
system_des = c2d(system , Ts) ; % discrete system transfer function
A = system_des.den ;  A = cell2mat(A) ;
B = system_des.num ;  B = cell2mat(B) ;  B = B(2:end); % remove extra zero
C = 1;

Np = 5 ; % prediction horizon
Nu = 1 ; % control horizon
Alpha = 1 ; % constant
na = numel(A)-1  ;
nb = numel(B) -1 ;
nc = numel(C) -1 ;
delta = [1    -1] ; % constant

[PHI , PI , OMEGA , E , F , G] = MPC_Matrix(A , B , delta , Np , Nu) ; % generate MPC matrixes
Ahat = conv(A , delta) ;
nah = numel(Ahat) -1 ;
W = [ 0*ones(floor(int16(numel(t)+Np)/2) , 1) ; 100*ones(floor(int16(numel(t)+Np)/2), 1) ]; % setpoint vector
y = zeros(size(t)) ;
u = zeros(size(t)) ;
du =  zeros(size(t)) ;
Uminus = zeros(nb , 1) ;
Yminus = zeros(na+1 , 1) ;

%% Mail Loop
for i = na+1:numel(t)
    y(i) = -A(2:end) * y(i-1:-1:i-na) + B * u(i-1:-1:i-na); % simulated data, in MCU implemenattion sensor data
    Uminus = u(i-1:-1:i-nb) -  u(i-2:-1:i-nb-1) ;          % data will be replaced, i.e. real RPM measurement
    Yminus = y(i:-1:i-na) ;                                % by encoder attached to DC motor
    freeReponse = PI * Uminus + PHI * Yminus ;
    dU = (OMEGA' * OMEGA +  Alpha * eye(size(OMEGA , 2)))\(OMEGA'*(W(i+1:i+Np)-freeReponse)); % control law
    du(i) = dU(1);
    u(i) = u(i-1) + du(i) ;  % update input signal to system & loop until end of vectot t
                             % this will go into the DC motor as voltage
end

%% plot Results
figure(1) ;
subplot(2,2,[1 2] ) ;
plot( t , W(1:numel(t)) , t , y , 'LineWidth' , 2) ; hold on
xlabel('Time  (Second)') ;
ylabel('Amplitude  Y') ;
grid on
title('GPC algorithm - output response vs sepoint') ;

figure(1) ;
subplot(2,2,3) ;
plot( t  , du , 'LineWidth' , 2) ; hold on
xlabel('Time  (Second)') ;
ylabel('Amplitude  dU') ;
grid on
title('Increment Effort Control') ;

figure(1) ;
subplot(2,2,4) ;
plot( t  , u , 'LineWidth' , 2) ; hold on
xlabel('Time  (Second)') ;
ylabel('Amplitude  U') ;
grid on
title('Effort Control') ;
```

Now, look at the MATLAB code result. Input is a step of 100 RPM; we ask the DC motor to go with 100 RPM. The output is depicted in Figure 12.

Figure 12

MATLAB code simulation outputs



As we see in Figure 12, the setpoint changes from 0 RPM to 100 RPM. Accordingly, output has changed from 0 to 100 RPM in around 150 ms (settling time). Input to process is approximately 4.17 volts. It is consistent with a gain of a transfer function of 24. That is 24*4.17=100.08 RPM as the setpoint is 100 RPM. *Prediction horizon and control horizon (Np and Nu in the above code)* are MPC parameters, in which the user can tweak the output response to a more robust, aggressive, faster, or slower style. Adjusting these two parameters is simple and won't be changed if the setpoint changes (contrary to PIDs). In this project, the range for prediction horizon N_p is between 2 to 10, and the range for the control horizon N_u is somewhere between 1 to 5. The latter parameter must be less than the former parameter value. Every application and process need its reasonable range of parameters value, which is determined by trial and error but is simple in the MPC.

4. C coding to implement our choices of hardware and run MPC experiment

In Figure 13, four significant parts of this work are shown. ARM 4 processor-based microcontroller board, 12 V DC motor with an encoder attached to it, rotary encoder switch for setting desired speed, and the bridge PWM motor driver. These parts and C source code residing inside the flash memory of MCU demonstrate a negative feedback loop depicted in Figure 4. A better view of the Nucleo board is illustrated in Figure 14.

Figure 13

All components need to build the DC motor speed MCU controller prototype

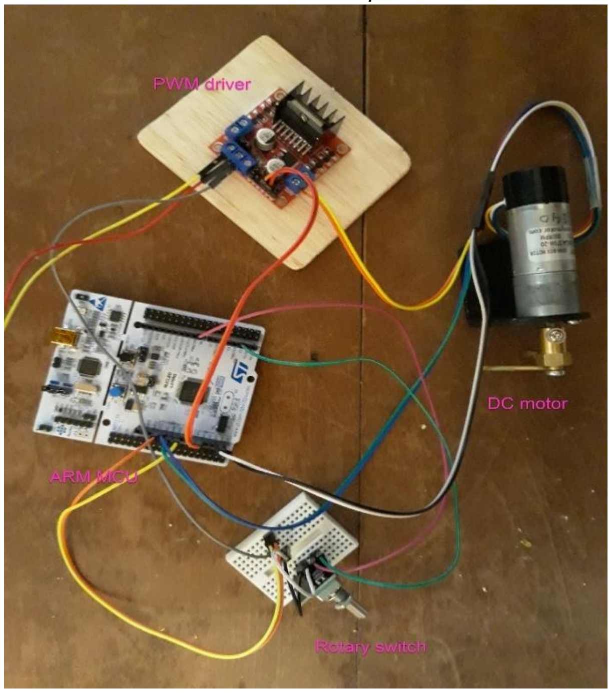
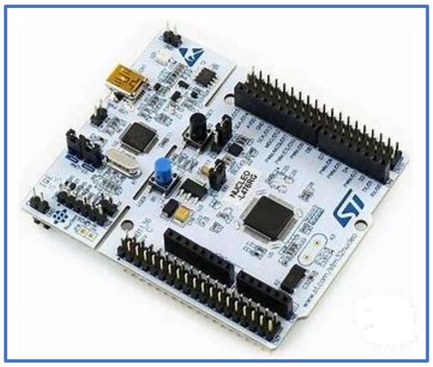


Figure 14

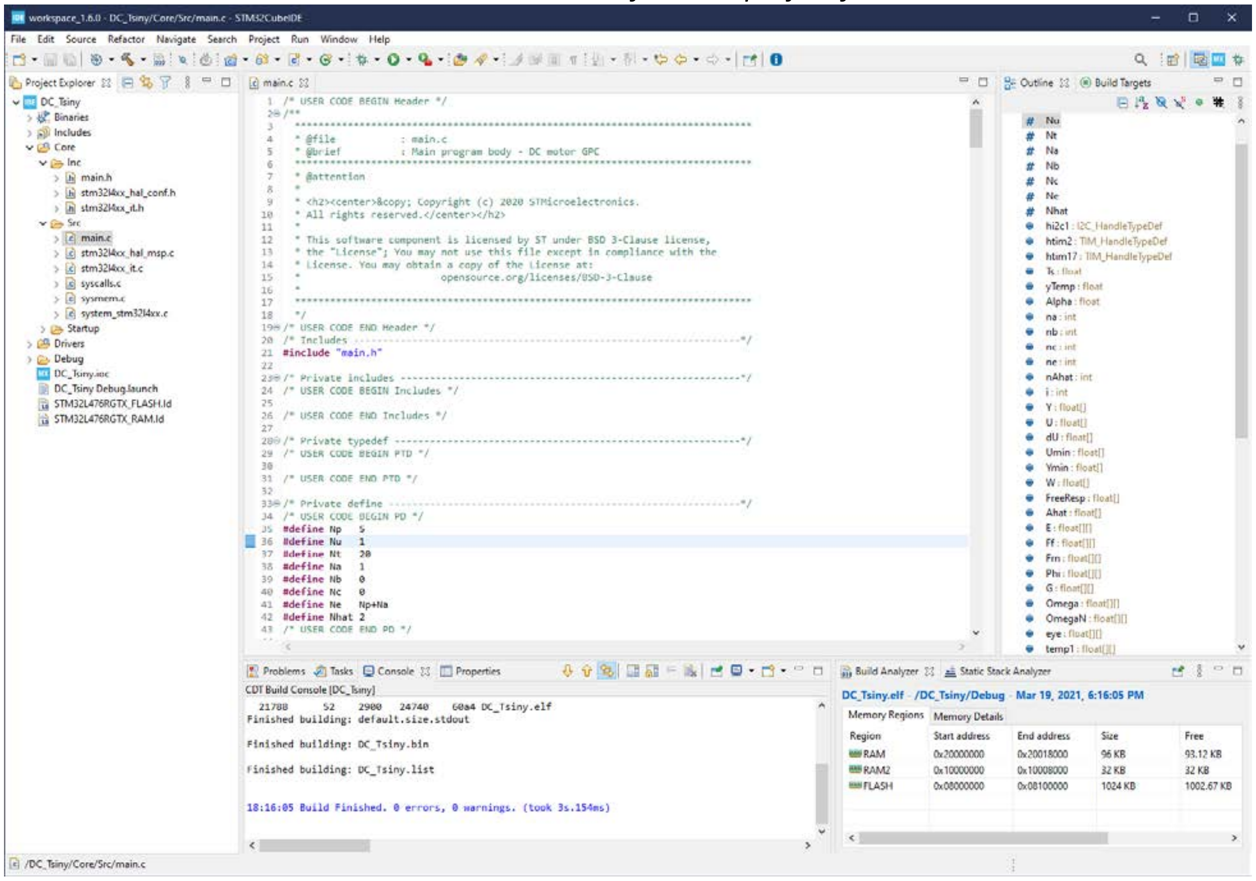
STM32 Nucleo board used in this work



The first page of the IDE project file is depicted in Figure 15. It is a C language with predefined functions that starts with the prefix HAL_ These functions are built-in and are input/output commands to use the hardware board. Figure 15 shows the debugging mode of the MPC project file. In the zoomed-in part shown in Figure 16, you will see the RPM is 100.5, and U[i] is 4.3 volts (input voltage to DC motor). As expected, both values from the simulation and the HIL result are approximately 100 RPM. We obtained 4.17 volts from the simulation result and 4.3 volts in lab measurement. Since the adapter (ac-dc) for PWM (Pulse Width Modulation) circuit board is at 13 volts and not really at 12 volts, there are 4.3-4.17=0.13 volts on top of 4.17 volts for actual measurement. Also, 0.075 is scale factor since the 12 volts PWM span is divided to 160 parts. Therefore, each increment will be 12/160=0.075. Increased PWM span to a higher value and causes the motor to rotate more softly and smoothly since the input of the DC motor signal is stepwise and not a continuous analog signal. Almost 160 parts cover 0 to 12 volts in a step of 0.075 volts (75 milli-volts), which results in a smooth DC motor rotation.

Figure 15

ST IDE environment for MPC project file


workspace_1.6.0 - DC_Tsiny/Core/Src/main.c - STM32CubeIDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
main.c
Outline
Build Targets
Problems Tasks Console Properties
Build Analyzer Static Stack Analyzer
/DC_Tsiny/Core/Src/main.c

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body - DC motor GPC
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define Np   5
#define Nu   1
#define Nt   20
#define Na   1
#define Nb   0
#define Nc   0
#define Ne   Np+Na
#define Nhat 2
/* USER CODE END PD */
```

CDT Build Console [DC_Tsiny]

```
   21788      52    2900   24740    60a4 DC_Tsiny.elf
Finished building: default.size.stdout

Finished building: DC_Tsiny.bin

Finished building: DC_Tsiny.list


18:16:05 Build Finished. 0 errors, 0 warnings. (took 3s.154ms)
```

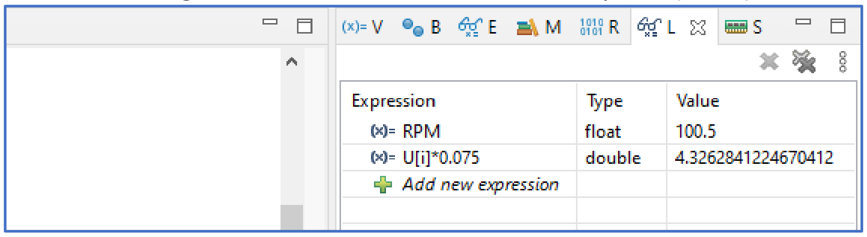
DC_Tsiny.elf - /DC_Tsiny/Debug - Mar 19, 2021, 6:16:05 PM

Memory Regions | Memory Details

Region	Start address	End address	Size	Free
RAM	0x20000000	0x20018000	96 KB	93.12 KB
RAM2	0x10000000	0x10008000	32 KB	32 KB
FLASH	0x08000000	0x08100000	1024 KB	1002.67 KB

Figure 16

Debug screenshot to show DC motor speed (RPM)



Expression	Type	Value
(x)= RPM	float	100.5
(x)= U[i]*0.075	double	4.3262841224670412
Add new expression		

Infrared RPM measurement is shown in Figure 17. As we can see, the reading is 97.8 RPM.

Figure 17

Infrared RPM measurement of DC motor

A summary of the steps taken in the study part of this work are:

- Simulation of one negative feedback loop using Simulink and LabVIEW
- Mathematical representation of predictive controller for a linear and non-constrained system
- Making a MATLAB code version of the above MPC for the DC motor model
- Verification of MATLAB code with prior simulation results
- Scripting C code for ARM-based M4 MCU
- Run hardware-in-loop (HIL) version
- Compare the actual measurement result of HIL and previous MATLAB code simulation

Analysis and Results

In this work, a DC motor speed controller based on MPC was built, and experiments were run to compare actual obtained values to the simulation results. Table 3 shows numerical results from simulation and actual measurements.

Table 3

Summary of speed measurements

	MATLAB simulation	Embedded C code	Infrared measurement
Speed	100 RPM	100.5 RPM	97.8 RPM

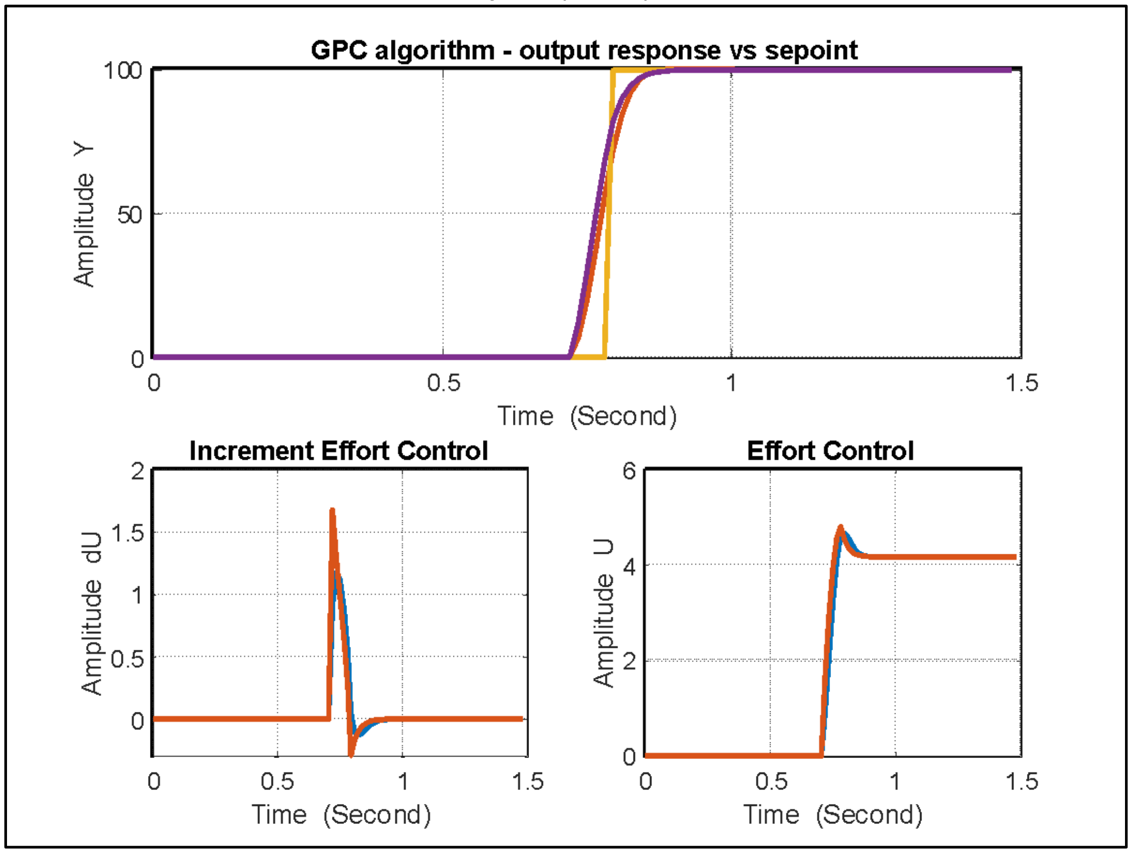
We can conclude that the prototype is functioning correctly. 100.5 RPM in HIL debug mode of IDE is achieved (Figure 16), as well as 97.8 RPM by infrared RPM measurement (Figure 17) compared to 100 in MATLAB simulation (Figure 12). While the setpoint is 100, all the above results are close to the setpoint. We have checked the speed from almost as low as a few RPM to the maximum RPM, 300. The results from simulation and infrared measurements were consistent.

At this moment, a reasonable question would be: how different would your process output be if the process had to be controlled by PID, and not MPC? The answer is "not significantly different" for this project. Every MPC solution optimizes a predefined cost function (Seborg, Edgar, & Mellichamp, 2004). In our MATLAB code, optimization is included with a variable Alpha=1. By increasing Alpha, the amount of energy that is taken from the input will be increased (the minimized cost function is $J=(W-Y)^t*(W-Y)+Alpha*u^t*u$, where W is setpoint, Y is output, and u is input. As you see Alpha weights the u^2 which is the input energy, we can have another new variable that weighs W-Y variable which means weighing differences between output and setpoint). Depending on the process, Alpha may have a significant or negligible effect on the output response. The range of Alpha is between a few hundredths to several thousand (unitless), Alpha equal to one is default value or no weighing on input contribution for cost-function optimization.

Figure 18 shows simulation outputs for Alpha equal to 1 and 1000. As we can see, the differences are minor.

Figure 18

Simulation results for Alpha equal to 1 and 1000



When there is no constraint, process limitation, cost function, and optimization in the SISO system, then PID response and MPC response are practically the same. The significant differences (or MPC advantages in this work) are:

- Simple tuning, PID needs three parameters while MPC requires two parameters to be adjusted. Other MPC parameters could be optimization factors (like Alpha).
- PID output will saturate in some conditions, but MPC controls input/output limits (constraints). In this work, in the beginning, when the motor is at 0 RPM, and we choose

300 RPM for the setpoint, the PID output goes to 12 volts and becomes saturated for a while until the speed of the motor goes high enough.

- PID cannot handle significant and variable delays, while MPC can make it even with a considerable system time constant (Olaru & Benlaoukli, 2008).
- MPC output characteristics like settling time, rise time, overshoot, and peak time don't change if the setpoint changes, but output characteristics change slightly in PIDs every time you change the setpoint.

This MPC implemented in this work has the above four advantages over a PID controller. The prototype built here can easily be applied to any other industrial, small-to-large DC motor if the static voltage levels are compatible. The MCU board has 0 to 3.3 volts input signal level and can drive an H-bridge PWM driver module. For higher voltage and current driving capability, we can switch to other available modules. The L298 driver module was used in this project, and for a higher fixed-voltage-current match to a larger DC motor, another module such as HiLetgo BTS7960 43A High Power Motor Driver can be used as shown in Table 4.

Table 4

Two driver modules for the current application and future high power DC motor "HiLetgo BTS7960 43A High Power Motor Driver Module"

Logic voltage: 5V Logic current 0 mA - 36 mA Storage Temperature: -20 ℃ to ℃ to +135 Operating mode: H-bridge driver (dual) Drive voltage: 5V-35V Drive current: 2A (MAX single bridge) Maximum power: 25W	
Input voltage: 6V-27V Maximum current: 43A Input level: 3.3-5V Control method: PWM or level Duty cycle: 0-100%	

Selected MCU in this work has enough core frequency, SRAM, and an adequate number of programmable inputs-outputs. The DC motor has a range of 0-12 volts DC input and 0 to 300 RPM output. The method of RPM measurement is hardware interrupt with a frequency of around 70 times per second. To work with a higher RPM DC motor, we need to increase interrupt occurrence per second or use timers instead.

Limitations and future work

This work is the first step of a series of MPC design steps discussed in the purpose section. There is no limitation to this work; if we know the process transfer function, we can use this work. For clarification, this project covers:

- Linear model with known transfer function
- Non-constrain process. It means that there is no minimum or maximum for the process variable value. It will be between ground to VCC value.
- The above code is pertinent to only the single-input single-output process.
- Process delay is not modeled

Future work could be an MPC that runs on MIMO systems, moreover, systems with nonlinearity.

Conclusions

MPC controllers are based on having a process model or system identification in terms of the transfer function. In most cases, when the process model is not obtained, we use traditional controllers like PIDs. The main advantage of having a process model-based controller is easy tuning. In MPC, we ended up with two parameters: control horizon and prediction horizon, instead of three parameters in PID. DC motor speed has been aimed and shown that actual measurement and simulation results are the same. C code for MPC works effectively, and a comparison between simulation and measurement is shown in Table 3. Applications of microcontrollers have been increased considerably, and there is one in each electrical equipment to replace analog PIDs with MPC.

References

A. Bemporad, "Model Predictive Control Design: New Trends and Tools," Proceedings of the 45th IEEE Conference on Decision and Control, San Diego, CA, USA, 2006, pp. 6678-6683, doi: 10.1109/CDC.2006.377490.

Åström, K., & Hägglund, T. (1995). PID controllers - THEORY, design, and tuning (2nd Edition). Retrieved March 20, 2021, from https://www.amazon.com/PID-Controllers-Theory-Design-Tuning/dp/1556175167

Chen, X., Heidarinejad, M., Liu, J., & Christofides, P. D. (2012). Distributed economic MPC: Application to a nonlinear chemical process network. *Journal of Process Control*, *22*(4), 689-699.

Chikkula, Y., Lee, J. H., & Ogunnaike, B. A. (n.d.). Robust model predictive control of nonlinear systems using input-output models. *Proceedings of 1995 American Control Conference - ACC'95*. https://doi.org/10.1109/acc.1995.531361

Darby, M. L., & Nikolaou, M. (2012). MPC: Current practice and challenges. *Control Engineering Practice*, *20*(4), 328-342.

Ibrahim, D. (2007). *Microcontroller based applied digital control*. Chichester: Wiley.

Jibril, M. (n.d.). Comparison of PID and MPC controllers for continues stirred tank reactor (CSTR) concentration control. – *philarchive.org*. Retrieved September 28, 2021, from https://philarchive.org/archive/JIBCOP.

Khaled, N., & Pattel, B. (2018). *Practical design and application of model predictive control: MPC for MATLAB® and Simulink® users*. Elsevier Science & Technology.

Ling, K. V., Wu, B. F., & Maciejowski, J. M. (2008). Embedded model predictive control (MPC) using a FPGA. *IFAC Proceedings Volumes*, *41*(2), 15250-15255.

Ljung, L. (2012). *System identification: Theory for the user*. Prentice Hall PTR.

Seborg, D. E., Edgar, T. F., & Mellichamp, D. A. (2004). *Process dynamics and control*. Hoboken, NJ: Wiley.

Singh, R. (2018). Model-based control system design and evaluation for continuous tablet manufacturing processes (via direct compaction, via roller compaction, via wet granulation). *Computer Aided Chemical Engineering*, 317–351. https://doi.org/10.1016/b978-0-444-63963-9.00013-0

Sivaram, Aung, K. T., Jadhav, A., Lynch, J., & Mahtsentu, A. (2021, March 4). *What is PID Controller ? - Instrumentation Tools*. Inst Tools. http://www.instrumentationtools.com/what-is-pid-controller/.

Sorin Olaru, Hichem Benlaoukli. MPC for systems with variable time-delay - robust positive invariant set approximations. (2008). *Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics Service*. https://doi.org/10.5220/0001501001770182