Lab Manual for EE380 (Control Lab)

Department of Electrical Engineering, IIT Kanpur

Manavaalan Gunasekaran and Ramprasad Potluri

Lab Manual Version: September 10, 2013

New in this version: Experiments 1, 2, 3 shortened;
Experiment 9 added.

Contents

Preface

Here we describe our considerations in designing the control systems laboratory component of EE 380 (course title "Electrical Engineering Laboratory") around an apparently simple DC motor control testbed.

0.1 Skills the control experiments need to impart

EE380 is a four credit hour laboratory course. Control systems constitutes one-third of this course. Given that we currently have only one control systems course active at the UG level at IITK, and that the one-third of EE380 is the only exposure the students have to a control systems laboratory at IITK, what do we want the students to learn from this brief exposure to the controls lab?

Here is one answer: In addition to helping the students practice paper-based or PC-based design techniques, most of which they may have seen in their lecture course on control systems, we believe that controls experiments need to help the students acquire the following skills associated with converting the paper-based or PC-based design into a practical system:

1. Ability to identify the hardware and software that are needed in a basic control system.

2. Ability to make this hardware and software work together.

3. Ability to debug small errors that may appear during practical implementation.

This knowledge comes only through at least a few weeks of work on problems, all of which may be related to one or two hardware setups that are not — and do not look — complex.

Overall, the lab experiments need to give the student confidence enough to say, "I have practical experience with implementing control systems in addition to designing and simulating them".

0.2 Past status of Control Systems Laboratory

Up to the August – December semester of 2008 EE380 had 4 sections of up to 24 students. Each section was divided into 6 groups of up to 4 students.

0.2.1 Logistical challenges

1. Six different experiments were done concurrently during each lab session with support from two TAs per session. Therefore, each TA and tutor needed to know all the 6 experiments every week, thus putting pressure on them.

   Thus, in any given week, the TAs expended more effort than if they were all preparing for the same experiment.

2. Also, with increased student intake (with up to 30 students per section), under that model, we would have had cacophony in the lab with everyone speaking about a different experiment.

3. With increased student intake, multiplying the then existing set of experiments would have been expensive. It would have been expensive to increase the number of inverted pendulums, or the number of ball and beam setups, or even the number of DAQ cards from NI. An inverted pendulum or a ball-beam setup comes for about Rupees Five Lakh each.

0.2.2 Solution to these challenges

The solution is *for all the students, TAs, and tutors to do the same experiment in a given week.* This model exists in the ESO210 labs, for example. It has the following advantages:

1. We will need only 2 – 3 TAs per section.

2. The students, TAs, and tutors will generate more knowledge than if they were all working on different experiments.

3. It will be easier for the entire class as everybody is talking about the same thing in a given week.

0.3 Planning for the future

0.3.1 Models for the experiments

We may have two models for the control experiments:

Model 1 The student sees one experimental setup in each experiment (e.g., magnetic levitation, dc motor control, ball and beam, inverted pendulum, etc.). The student designs a controller for the given system based on a mathematical model that was provided by the control system's manufacturer,

and inputs the values of the controller's parameters into a convenient interface provided on the control system. The control system itself has been built by someone else and is almost a black box to the student.

Pro: This way, the student becomes acquainted with the various control experimental setups that are available in the market, and the real-life system each of these setups models. But, the student could have learned this from `www.youtube.com` too.

Pro: The students may learn that a control system works differently in practice than on paper.

Pro: This kind of an experiment impresses upon the student the wide applicability of control systems theory.

Con: The student does not see the hardware innards of the control system, nor does he/she talk to anybody that has actually built this setup and could share his/her experience building it.

Model 2 The student works with only one or two experimental setups through the semester.

Pro: The student solves many different problems associated with each setup. This way, the student can learn how a practical control system is actually built after the paper-/PC- based design and simulation.

The pros in Model 1 are not significant enough for the student to spend a semester in the EE380 labs. On the other hand, the pro of Model 2 is. We recommend Model 2 as it is in consonance with Sections 0.1 and 0.2.2.

0.3.2 Suggested new set of experiments

We recommend a phased introduction of Model 2 described in the previous subsection. Towards this end, we suggest that in the first two years of introduction of this new plan, the students will perform single-loop experiments that only involve the control of a DC motor, and design and simulation using MATLAB/Octave/Scilab.

The DC motor control experimental setup offers rich possibilities for learning the practical aspects of control systems design and implementation. Quanser has a DC motor control kit with a user manual that lists at least 6–7 experiments[1]. We could borrow ideas from that list too apart from using the experiments that we have already designed.

In the July — December semester of 2009, we introduced 4 new experiments involving control of DC motor. Thus, we already have experience with these new experiments.

For additional details, please see the paper [1].

[1] `http://www.quanser.com/english/downloads/products/Mechatronics/QET%20PIS_031708.pdf`

Contributions to the lab

Late 2008 Dr. Ramprasad Potluri conceptualized the lab as outlined in the Preface.

Early 2009 Mr. Manavaalan Gunasekaran suggested development of the dsPIC boards for a 4WD4WS vehicle that we plan to build in the Networked Control Systems Laboratory.

June – July 2009 Ms. R. Sirisha and Mr. Yash Pant, 4th year BTech students, College of Engineering, Roorkee, designed, built, tested, and documented the first prototype of the dsPIC board.

Mid-June 2009 Dr. Potluri was asked to teach CS-EE380-Fall2009.

July 2009 Mr. Manavaalan Gunasekaran improved the boards and implemented the first four experiments.

Fall 2009 The dsPIC boards were put to use in CS-EE380-Fall2009.

Dr. Adrish Banerjee gave valuable feedback and encouragement from his stint as a tutor for this lab.

Funds were announced by IITK's capacity expansion program (CEP) (2009) to set up the pmdc motor control-based experiments in a new control systems laboratory.

Summer 2010 Mr. Mohit Gupta, a 4th year BTech student of Manipal Institute of Technology, helped multiply the dsPIC board. He also helped make a few ergonomic improvements.

Summer 2009 & summer 2010 The dsPIC boards were fabricated in the PCB Lab of the EE department under Mr. Kole's supervision. Mr. Kole suggested several ergonomic improvements.

Summer 2010 Mr. Uday Mazumdar, in-charge of the Control Systems Lab had the mechanical part of the setup built and assembled. He also supervised setting the lab up in the new room (WL216).

Mr. Sripal of the Basic Electronics Lab populated 21 of the dsPIC boards.

Mr. Harishankar populated the H-bridge boards.

At every stage beginning the CEP, the lab received support from the then Head of the department, Prof. Ajit Kumar Chaturvedi.

July 2011 Mr. Uday Mazumdar assumed the responsibility of trouble-shooting the hardware problems that arise on the boards.

January 2012 A paper [1] that describes this laboratory in details was published.

September 2013 A modified version of Experiment 4 was introduced as Experiment 9, based on Mr. Kumar Saurav's paper [2]. This paper was a result of Mr. Kumar Saurav's work during May 2012 – June 2013.

Chapter 1

The experimental setup

1.1 Introduction

The block diagram of the setup is as shown in Figure 1.1. A dsPIC30F4012 micro-controller from Microchip (`www.microchip.com`) is used to house the P/PI/PID or any other controller that we will design. An H-bridge two-quadrant DC chopper board built around an L298 dual motor driver chip by Solarbotics (`www.solarbotics.com`) is used to drive the DC servo motor.

One PWM signal, one direction control signal, and one enable signal are required to control the motor in two-quadrant operation using L298. The following section provides a short description about the dsPIC30F4012 micro-controller and its programming to see how these three signals are generated.

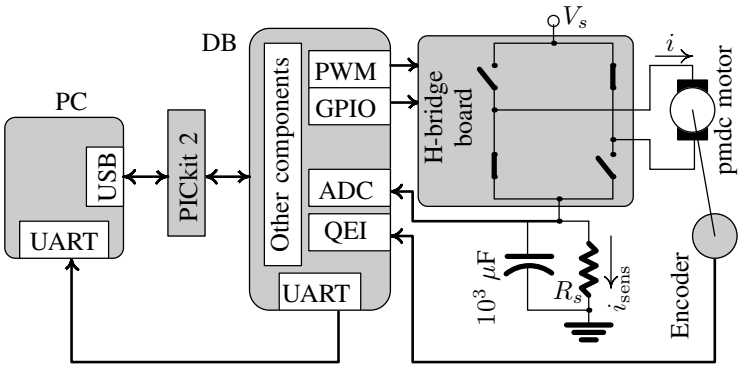


Figure 1.1: Block diagram of the setup. DB stands for *dsPIC board*.

1.2 Microcontroller dsPIC30F4012

The dsPIC30F4012 is a 16-bit microcontroller that Microchip calls a *digital signal controller*. dsPIC30F4012 has been optimized for motor control application.

Here is a brief description of some of its features used in our setup.

1.2.1 Timer Module

We use Timer-1, one of the five timers of the timer module, to generate an interrupt at each sampling instant. When the interrupt occurs, dsPIC executes the Timer-1's interrupt service routine (ISR). We have written this ISR in `main-prog.c` to perform the tasks shown in Figure 1.4 and further elaborated in the timing diagram of Figure 1.5.

We have set the internal clock frequency F_{CY} to one fourth the oscillator frequency F_{OSC}, that is, ($F_{CY} = F_{OSC}/4$). In practice, $F_{CY} \leq F_{OSC}/4$ in dsPIC30F4012. We use this F_{CY} in the timer module. For a sampling time T_s in seconds, the value $PR1$ in the period register is calculated as follows:

$$PR1 = \frac{T_s}{T_{CY}} = F_{CY}T_s = \frac{F_{OSC}T_s}{4}$$

In our dsPIC board F_{OSC} is 29.492 MHz.

Note 1.1. *An adequate sampling period T_s would be one within which the ISR of Timer 1 completes execution. Therefore, to determine a T_s that would be adequate for our purpose, we determine as follows the time that the ISR needs to execute. We set the GPIO pin E8 ($LATEbits.LATE8 = 1$) when the timer interrupt occurs (code starts to execute at the sampling instant), and reset this pin ($LATEbits.LATE8 = 0$) when the ISR completes execution. The output from this pin from set to reset is a pulse and can be seen on an oscilloscope. The duration of this pulse is the execution time of the code. If this duration is not less than T_s, we need to increase T_s.*

In our experiments, $T_s = 2$ ms was found to be adequate.

1.2.2 Pulse Width Modulation (PWM) Module

The PWM module is used to generate a PWM signal with duty ratio computed from the controller output. We have used $PWM1L$ to drive the H-bridge circuit. If the controller output is u then the duty ratio is $D = u/V_s$, where V_s is the power supply voltage to the motor driver in volts. To generate a PWM with a frequency F_{PWM} in Hz and duty ratio D, the settings of the PWM module are

$$PTPER = F_{CY}/F_{PWM} - 1, \qquad PDC1 = 2D(PTPER + 1)$$

In our experiments $F_{PWM} = 50$ kHz.

1.2.3 Quadrature Encoder Interface (QEI) Module

It is used to interface the motor speed encoder to calculate the speed. Figure 1.2 explains the working of the quadrature encoder. The signals of the QEI are

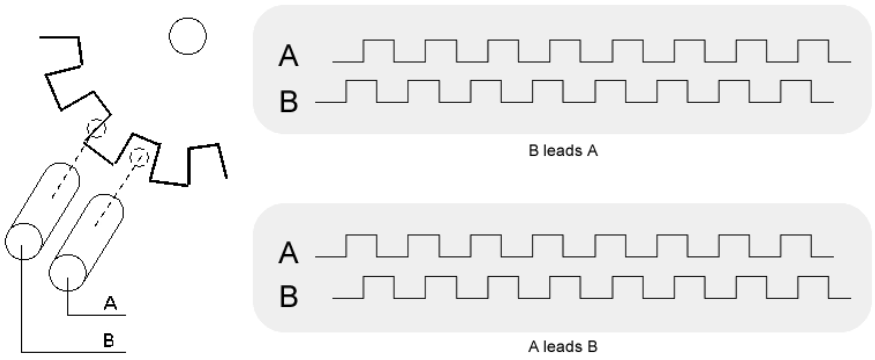


Figure 1.2: How the quadrature encoder works. The encoder comprises three parts: an opaque disc with slots of equal width cut at equals intervals along its circumference, two sources of light on one side of the disc, and two receivers of light on the other side of the disc. The sources and receivers are mapped one-to-one. The signals from the receivers are labeled A and B. Assume that light being blocked by the disc represents a logical 0, and light being passed by the disc represents a logical 1. B leads A when the disc rotates clock wise and A leads B when the disc rotates anti-clockwise.

shown in Figure 1.3. The QEI is configured in x2 mode ($QEICON<10:8>=$ 101). In this mode both edges — rising and falling — of the phase-A signal cause the position counter to change value (increment or decrement). The phase-B signal is utilized for the determination of the position counter's direction (increment or decrement).

The value of the count, which — for each period T_s — is twice the number of pulses from the encoder, is accessible through a register $POSCNT$.

If the encoder has a resolution of C_T counts per turn, then the speed ω in rad/sec is given by

$$\omega = \frac{2\pi}{T_s}\frac{POSCNT}{2C_T}$$

1.2.4 Universal Asynchronous Receiver Transmitter (UART) Module

This is used to communicate with Personal Computer (PC). We use this to send the data of speed and controller output (which is the voltage we wish the H-bridge to apply to the motor) to the PC for plotting. UART is configured with one stop bit and the register $U1BRG$ is used to set the baud rate. Let B_R be the baud rate then the register value is given by

$$U1BRG = \frac{F_{CY}}{16B_R} - 1$$

In our program, the baud rate is 115200. This is the maximum possible baud rate for our dsPIC board with 29.492 MHz oscillator and its corresponding value for the U1BRG register is 0x0003.

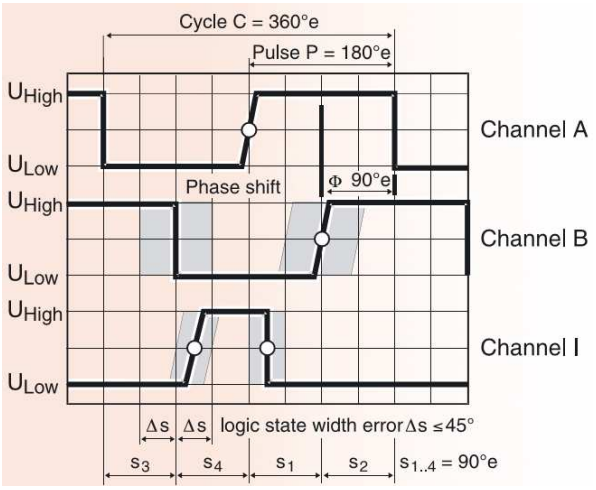


Figure 1.3: Signals of the QEI. Figure copied from Maxon Motor.

1.2.5 GPIO Pins

The GPIO pin *D*0 is used as a digital output to change the polarity of the voltage output by the H-bridge (to change the direction of the rotation of the motor). The data direction register (TRISx) determines whether the pin is an input (1) or an output (0). Reading or writing the latch is done by using LATx.

See the section titled `pwm_control function` in the file `setting-prog.h` for further details.

In our case, TRISD = 0 (Port D is configured as output port).

1.2.6 Analog to Digital Convertor (ADC)

Pins AN(0-2) are configured as analog inputs by using register ADPCFG. Auto conversion mode is used (SSRC(2:0) = 111). The analog input to be converted is selected using ADCHS register (CH0SA(3:0) bits e.g. 000 for AN0, 001 for AN1). In settings program AN0 is alone selected for conversion.

1.3 Choice of sampling interval

As our microcontroller does not have enough on-chip memory to hold the data we generate during each control run of the setup, we communicate to the PC the data as and when it is generated, which means within each sampling interval. In the experiments we do in EE380, this data is that of position, speed, armature voltage, and current.

If we wish to communicate only two of these variables to the PC, then it seems that $T_s = 0.002$ s may be adquate. In our earlier trials, when we tried to communicate three variables to the PC, we found that, while a sampling interval of

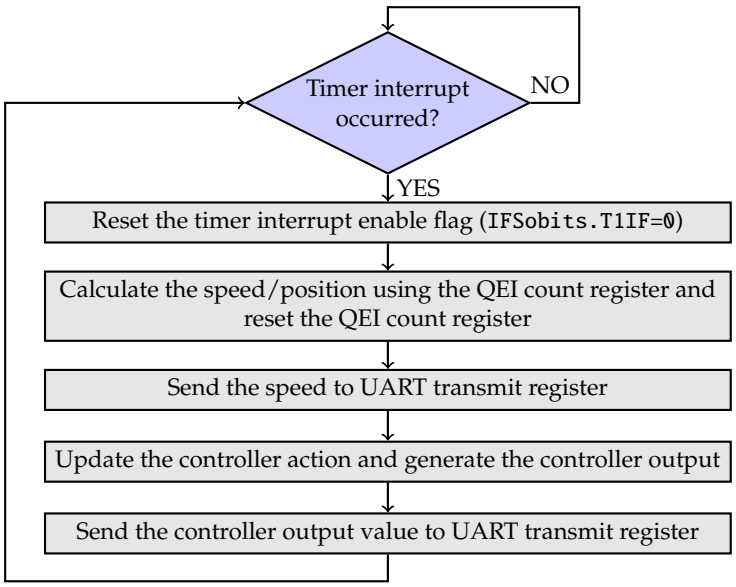


Figure 1.4: Flow chart of the ISR in `main-prog.c`.

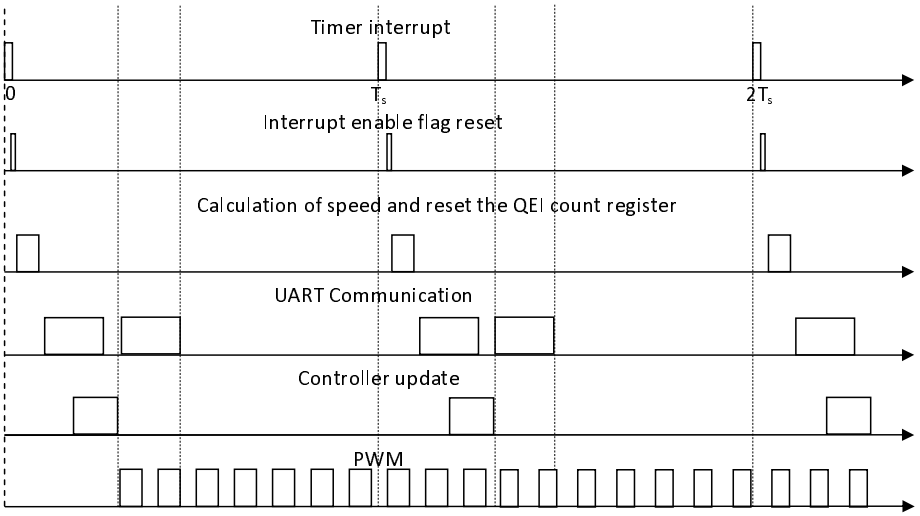


Figure 1.5: Timing diagram for the tasks that the ISR implements.

0.002 s was not adequate, even a sampling interval of 0.003 s, which seems to be a reasonable choice, was inadequate occassionally. Here, by *inadequate* we mean that the log file created by `terminal.exe` contained non-numeric data where it should have contained numeric data. We hypothesize that this inadequacy may be due to the non-realtime nature of how MS Windows may handle UART communication given that we have not made a provision for handshaking signals (see Subsection 1.7.2). However, this hypothesis needs testing.

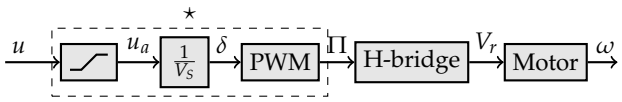


Figure 1.6: Diagram of the plant for which the controller is designed, used in all the experiments except the experiment on Ziegler-Nichols tuning. The part enclosed in the dashed box and labeled $\star$ is inside the dsPIC microcontroller, u_a is the numerical value of voltage applied to motor's armature, V_S is the source voltage applied to the H-bridge board, and δ is the duty ratio. The dsPIC outputs a PWM signal Π of duty ratio δ to the H-bridge board. The H-bridge outputs a variable-magnitude DC voltage V_r to the motor. The saturation block implements the operation $-0.8V_S \leq u_a \leq 0.8V_S$.

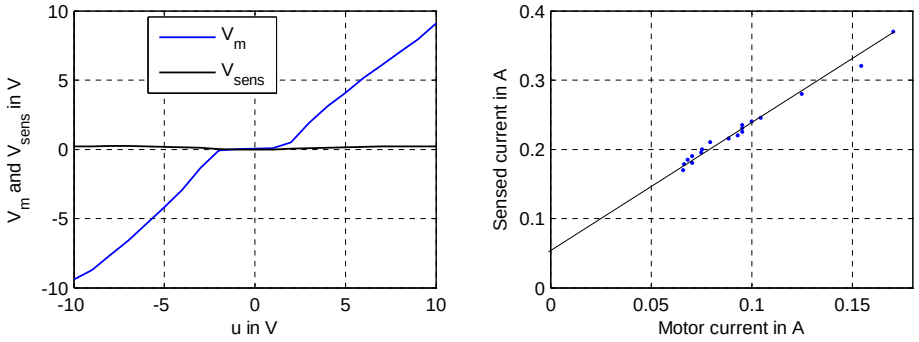


Figure 1.7: Left-fig: The blue curve is the voltage V_m (same as V_r in Figure 1.6) output by the H-bridge versus the input voltage u specified by the microcontroller; the black curve is the voltage across the sensing resistor ($R_s = 5\ \Omega$) versus u. The supply voltage is 12 V. Right-fig: The current i_{sens} through the sensing resistor versus actual motor current i.

1.4 Parameters of PMDC motor-gear-encoder unit

For data sheets of the motor, gear, and encoder, Google with the key words `A-max26 110963` (for motor), `GP26B 144036` (for gear), and `HEDS 5540 110511` (for encoder). These three items are from Maxon Motor (`www.maxonmotor.com`). The parameters of our PMDC motor are reproduced in Table 1.1.

1.5 Characteristics of the H-bridge board

This section is based on tests we performed on the plant of Figure 1.6.

Figure 1.7-left shows that the relationship between the microcontroller output u and the H-bridge output V_m is nonlinear. This relationship has a dead zone for $|u| \leq 2$ V, and is linear for $|u| > 2$ V. This characteristic concerns us in, for example, experiments 2, 3, and 4.

Figure 1.7-right shows the relation between the actual motor current i and the current i_{sens} sensed using the sensing resistor R_s for V_s constant and T_L varying. This data suggests that $i \approx 0.533 i_{\text{sens}} - 0.027$.

Table 1.1: Parameters of the DC servo motor.

Parameter	Value	Units
Nominal voltage	24	V
No load speed	8820	rpm
No load current	31.7	mA
Nominal speed	6730	rpm
Nominal torque	17.2	$\text{mN} \cdot \text{m}$
Nominal current	0.704	A
Stall torque	77.6	$\text{mN} \cdot \text{m}$
Starting current	3.04	A
Max. efficiency	79	%
Terminal resistance	7.9	Ω
Terminal inductance	0.770	mH
Torque constant	25.5	$\text{mN} \cdot \text{m/A}$
Speed constant	374	rpm/V
Rotor inertia	13.0	$\text{g} \cdot \text{cm}^2$
Gear mass inertia	0.4	$\text{g} \cdot \text{cm}^2$
Gear ratio	62:1	
Encoder: Counts per turn	500	

Using i_{sens} and the drop V_H in the H-bridge, we found that the H-bridge has a resistance R_H of about 27.5 ohm. V_H and R_H were determined as $V_H = u - V_m - V_{\text{sens}}$ and $R_H = V_H / i_{\text{sens}}$. V_m and V_{sens} were measured using a digital multimeter. I_{sens} was found as $i_{\text{sens}} = V_{\text{sens}} / R_s$.

The response of the motor to a step u of magnitude 5 V has a steady-state value $\omega_o = 150$ rad/sec. This response is divided by 5 and plotted in Figure 1.8. Also plotted in the same figure is the unit step response of the physics-based TF $K_m/(\tau_m s + 1)$ of the same motor. This TF was obtained as described in Section 2.3. The values of the parameters in this TF are directly or indirectly determined from Table 1.1 and converted into SI units. While K_T and J can be read off the table, and K_b — the back EMF constant — is the reciprocal of the speed constant, B and $R_\sum$ need a little thought.

Even though we included the H-bridge as a resistor, as described in Section 1.6, there is a mismatch in K_m by 3 units and time constant by 0.01 s. This mismatch needs investigation.

1.6 Calculation of B and armature resistance

$B\omega$ is the viscous friction torque in the bearings of the motor, and B is the coefficient of viscous friction in the bearings. The literature says that the viscous

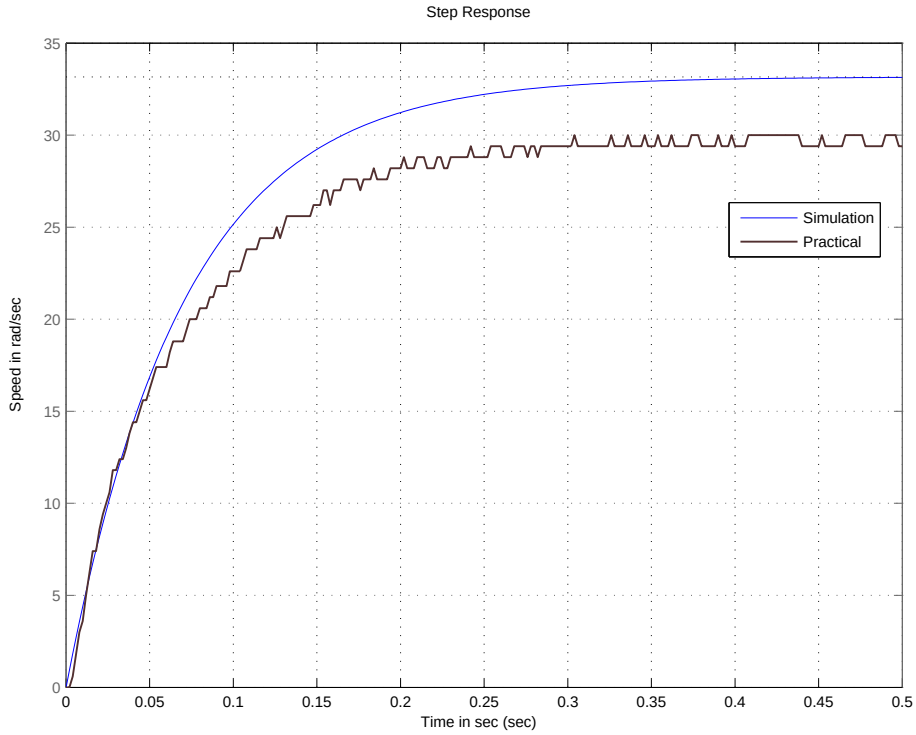


Figure 1.8: Step responses of physics-based TF (denoted *Simulation*) and actual plant (denoted *Practical*).

friction (as opposed to rolling friction, Coulomb friction, etc) is the dominant component of bearing friction.

When no load is applied to the motor shaft ($T_L = 0$) and the speed is steady ($\frac{d\omega}{dt} = 0$), the developed torque $K_t i$ needs to be enough to equal only $B\omega$.

In the same experiment that helped generate the practical curve of Figure 1.8, the no-load motor armature current was found to be $I_{mo} = 0.0173$. Also, we saw that $\omega_o = 150$ rad/sec. From the data sheet $K_t = 25.5$ mNm/A. Therefore, $B = K_t I_{mo}/\omega_o = 2.9374 \times 10^{-6}$ Nm/(rad/s).

The actual resistance in the motor armature circuit is $R_\sum = R_m + R_s + R_H$. The motor terminal resistance R_m equals 7.9 Ω. Thus, $R_\sum = 7.9 + 5 + 27.5 = 40.4$ Ω.

1.7 Programming

1.7.1 Writing from PC to dsPIC

MPLAB Integrated Development Environment (IDE) v.8.30 and the academic version of MPLAB C30 C compiler are used to create a '.hex' file from the project (C language program). This '.hex' file will be loaded to the

dsPIC30F4012 through PICKIT-2 using PICKIT2 v2.61 software. We wrote a code to configure all the modules that are necessary for our setup. In that code the controller part alone needs to be modified according to your own designed controller. The simple procedure to write a program into dsPIC is as follows:

- Run the MPLAB IDE and open the project file which we have provided you on the Desktop of your PC.
- In that project C-program file the controller code part should be modified to form your own designed controller.
- Save all and go to Project → Build All. Make sure that the PICkit 2 is chosen in Programmer → Select Programmer.
- Export the **.hex** file (File → Export). Remember the file name and location.
- Run the PICkit 2 Programmer software, choose the device family dsPIC30, import the '.hex' file, and write the program to dsPIC30F4012 by pressing the **Write** command button.

1.7.2 Reading the data from dsPIC to PC

Run the Hyper Terminal or Bray's Terminal.exe v.1.9b – 20040204, choose COM1 Port, set Baud rate to 115200, 8-data bits, none parity bit, 1-stop bit and none handshaking bit. Store the data in the text file and import the data to MATLAB. The odd indexed data are values of speed ($\omega(i * T_s) = \text{data}_i$; $i = 1, 3, 5, \cdots$) in rad/sec. The even indexed data are 100 times the controller output data ($v(i * T_s) = \text{data}_i/100$; $i = 2, 4, 6, \cdots$) in volts.

1.8 Program listings

1.8.1 main-prog.c

```
// main-prog.c.

#include<p30f4012.h>
#include "settings-prog.h"
_FOSC(CSW_FSCM_OFF & XT); // To use the external crystal
_FWDT(WDT_OFF); // To disable the watchdog timer

// split the data by decimal digits (0 - 9)  in 3 digit form
void send_data(int);

int AD_value(); // Declare the AD value reading function
double pulse;
float V_s, duty, u, error, speed, pos, T;
float Iest[2], West, wf, IV;
float  Fpwm, Kb, Ra, R;
float ac,bc,cc,dc;
float x;
float Is, IF, Ihat;
```

```
//
// Declare your variables here
//

void main()
{
  // Initialise your variables here

  wf = 50;       // Filter cut-off frequency in rad/sec
  error = 0.0;
  duty = 0.0;
  u = 0.0;       // initialise the controller o/p
  speed = 0.0;   // Rad/sec
  pos = 0.0;     // radians
  T = 0.002;     // Sampling time in sec
  Fpwm = 50;     // PWM Frequency in kHz
  V_s = 12.0;    // Power supply Voltage
  TRISD = 0;     // D port is configured as output port
  LATD = 1;      // used for direction control

  qei_set();     // Initialise QEI settings
  pwm_set(Fpwm); // Initialise PWM settings
  uart_set();    // Initialise UART settings
  AD_set();      // Initialise ADC settings
  timer1_set(T); // Initialise Timer-1 settings & start timer
  TRISEbits.TRISE8 = 0; // RE8 is configured as output

  // You need to use your own values of ac, bc, cc, dc below
  ac = 1-0.2*T; bc = T; cc = 0.29898; dc = 0.0151;
  R = 100; Ra = 28.7; Kb = 0.0255;
  // I[0] = 0; I[1] = 0; IF[0] = 0; IF[1] = 0;
  Is = 0; IF = 0; Ihat = 0;

  // Continue until stop the power
  for(;;);
}// End of main()

// Interrupt service routine (ISR) for interrupt from Timer1

void __attribute__((interrupt, no_auto_psv)) _T1Interrupt (void)
{
  IFS0bits.T1IF = 0; // Clear timer 1 interrupt flag

  // To calculate the execution time of the controller code make E8 = 1

  LATEbits.LATE8 = 1;
  uart_tx(9);

  // QEI count feedback
  // if motor is in anticlockwise direction,count goes down from FFFF.

  if(POSCNT > 0x8000)
  {
```

```
  pulse = 0xFFFF - POSCNT;
  pulse = - pulse;
}
else
  pulse = POSCNT;
POSCNT = 0; // Reset the QEI count

// Calculation of speed (rad/s)
// speed = 2*pi*( no of pulse/(2*500) )/T in (rad/sec)
speed = 6.2831853 * pulse/1000/T;
send_data(speed);  // Transmit the speed

// send_data(West);

// Calculation of position (rad)
// pos_current = pos_past + 2*pi*[no of pulse/(2*500)]/Rg in (rad)
// Rg = 62 gear ratio
// pos = pos + 6.2831853 * pulse/62000;
// send_data(pos*100);  // Transmit the position *100

// In the experiment where we input a sine wave that lies in the
// interval [0,5] V, and the speed reference is a sinusoid, enable
// the following two lines to give the reference input.
//
// R = AD_value(); // In signed mode, ADC maps [0,5] V to [-511,+511].
// R = 10.0*R/511; // R = 10*sin(w_in*t) rad/sec; 10 is max speed.
//
// R = R/511*10.0; // This is a working alternative to R = 10.0*R/511;
//
    // R = R/511*10 or R = 10*R/511 do not work as we intend. Need
    // to see how C language defines arithmetic operations.

// ---------------------- CAUTION -------------------------------
// By "amplitude", we mean half the peak-to-peak value,
// whereas the FG Scientific SM5078 means by "amplitude"
// the peak-to-peak value of the periodic signal. So, if
// you wish to apply a sinusoid/triangle/rectangle of
// peak-to-peak value 10 V, set the function generator
// through MODE --> AMP to 5 V.
// --------------------------------------------------------------

// West = (u-Ra*IF)/Kb;
// West = (u-Ra*Ihat)/Kb;

// Uncomment below 2 lines in experiments that use feedback of current.

// IV = AD_value();         // Read voltage across Rs=4.7ohm.
// IV = 5*(511 + IV)/1022;  // Convert signed to unsigned.

// Uncomment the following line to observe filtered current
//
// send_data(IF*1000);
//
// Why the 1000? We have observed in our trials that the current
```

```
  // is less than 1 A in magnitude. Irrespective of what the actual
  // values may be, for convenience, we send integers of at most 3
  // digits from the UART module. Therefore, when the current is
  // upto 0.999 in magnitude, we multiply its value by 1000. If we
  // find that it exceeds 0.999, then we may choose to multiply by
  // a number that will result in a product of at most 999.

  //  Ihat = (2.6)*IF;
  //  Is = IV/4.7;                    // Convert voltage to current.
  //  IF = (1-5.0*T)*IF + 5.0*T*Is;  // Low-pass filter.

  /*********** Start of your controller ****************/

  // error = R - West;
  // u = cc*x + dc*error;
  // x = ac*x + bc*error;

  /*-------------- Dead zone compensation-----------------*/
  /*
  if (u > 0)
    u = u + 2;
  else if (u <= 0)
    u = u - 2;
  */
  /*------------End of dead zone compensation-------------*/

  /*********** End of your controller ******************/

  // u = 5.0*AD_value()/511;

  // u=7; // For step input uncomment this  to provide step of 7

  if(u > 0.8 * V_s)
    u = 0.8 * V_s;           // Positive saturation
  else if(u < -0.8 * V_s)
    u = -0.8 * V_s;          // Negative saturation

  duty = u/V_s;
  pwm_con(duty);    // Update PWM using new duty ratio
  uart_tx(9);
  send_data(u*100); // Send 100 times control effort u.

  LATEbits.LATE8=0;
} // End of ISR of Timer 1


void send_data(int s_data)
{
  int s;
  if(s_data < 0)
  {
    // Send the negative sign (ASCII is 45)
    uart_tx(45);
    s_data = -1*s_data;
```

```
  }

  // Digit with the position value of 100
  s = s_data/100;
  uart_tx(s+48);

  // Digit with the position value of 10
  s_data = s_data - (s *100);
  s = s_data/10;
  uart_tx(s+48);

  // Digit with the position value of 1
  s_data = s_data - (s *10);
  uart_tx(s_data+48);
}// End of send_data()

int AD_value()
{
  int count, *ADC16Ptr, ADCValue = 0;  // clear value
  ADC16Ptr = &ADCBUF0;  // Initialize ADCBUF pointer
  ADCON1bits.ADON = 1;  // Turn ADC ON
  IFS0bits.ADIF = 0;  // Clear ADC interrupt flag
  ADCON1bits.ASAM = 1;  // Auto start sampling

  while (!IFS0bits.ADIF);  // Conversion done?
  ADCON1bits.ASAM = 0;   // If YES then stop sample/convert

  for (count = 0; count < 2; count++) // Average the 2 ADC value
    ADCValue = ADCValue + *ADC16Ptr++;
  ADCValue = ADCValue >> 1; //``>>'' represents ``shift by 1 to left''.
                            // Equivalent to ``divide by 2''.
  return(ADCValue);
}// End of AD_value()
```

1.8.2 settings-prog.h

```
// settings-prog.h: Settings for dsPIC30F4012 PCB with 29.492 MHz
// crystal oscillator

/*------------UART settings:---------------
  Fcy = Fosc/4 = 29492000/4 = 7373000Hz
  U1BRG = {Fcy/(16 * Baud_Rate) } - 1
  conf: 1 stop bit, 8 data bit, no parity
-----------------------------------------*/

#include<p30f4012.h>

void timer1_set(float); // Timer-1 settings
void qei_set();         // QEI settings
void pwm_set(int);      // PWM settings
void pwm_con(float);    // PWM control based on duty ratio
void uart_tx(int);      // UART data to be transfered to PC
void uart_set();        // UART settings
```

```
void AD_set();          // A-D settings

void pwm_con(float Duty)  // ******* pwm_control function
{
  int pdc;
  pdc = Duty * 2 *(PTPER + 1);
  if(pdc == 0)
  {
   // Shut down all the IR2110
   LATDbits.LATD1 = 1;
   LATDbits.LATD0 = 1;
  }
  else if(pdc < 0)
  {
   // Make sure IR2110-2 is shut down, and IR2110-1 is active
   LATDbits.LATD1 = 1;  // RD1 = 1, IR2110-2 is shut down
   LATDbits.LATD0 = 0;  // RD0 = 0, IR2110-1 is active
   pdc = 2*( PTPER+1) + pdc;
  }
  else if(pdc > 0)
  {
   // Make sure IR2110-1 is shut down, and IR2110-2 is active
   LATDbits.LATD0 = 1;  // RD0 = 1, IR2110-1 is shut down
   LATDbits.LATD1 = 0;  // RD1 = 0, IR2110-2 is active
  }
  PDC1 = pdc;
}

void timer1_set(float Ts)  // ******* Timer-1 settings
{
  IEC0bits.T1IE = 1; // Enable Timer-1 interrupt
  IFS0bits.T1IF = 0;// Clear Timer-1 interrupt flag to get next interrupt
  PR1 = 7373000*Ts;// No of clk (count) per controller sampling time
  TMR1 = 0;       // Initialize timer count
  T1CON = 0x8000;// Starts timer, Internal clock (Fosc/4), prescale 1:1
}

void qei_set()  // ******* QEI module settings
{
  ADPCFG = 0x0038;// Configure pins AN(3-5)/RB(3-5) to Digital I/O mode
                  // AN(0-2) pins are in Analog mode
  IEC2bits.QEIIE = 0; // Disable interrupt due to QEI
  IFS2bits.QEIIF = 0; // Clear the interrupt flag
  QEICON = 0;         // Default mode: QEI mode/timer off
  QEICONbits.QEIM= 5;
  DFLTCON = 0x0100;   // No filter operation
  POSCNT = 0;         // Initialize position of counter
  MAXCNT = 0xFFFF;    // set maxcount limit
}

void pwm_set(int F_pwm)  // ******* PWM module settings
{
  // PWM timer was enabled, 1:1 prescale Tcy, 1:1 Postscale,
  PTCON = 0x8000;        // PWM time base operates in free running mode
```

```
  PTPER = 7373/F_pwm - 1;// PWM Time Base Period Register (Period of PWM)
  // Note: PTPER = {Fcy/(Fpwm*PTMER_prescaler) - 1 }
  // Fcy =Fosc/4 = 7373000
  PWMCON1 = 0x0011; // PWM I/O pin pair is in complementary output mode
                    // PWM1L & PWM1H enabled; remaining are in I/O mode
  PDC1 = 0;         // Initially duty ratio is zero;
  OVDCON = 0x0303;  // Controlled by PWM module
  PTMR = 0x0000;    // PWM Time Base Register initialized
}

void uart_tx(int tx_data)  // ******* Transmit the Data through UART
{
  while(U1STAbits.UTXBF == 1)
  {
   // wait to the UART transmit buffer gets one emty space
  }
  // if(U1STAbits.UTXBF!=1) // If buffer is not full, transmit data
  U1TXREG=tx_data; // Transmit
}

void uart_set()  // ******* UART module settings
{
  U1MODE = 0x8400;       // 1-stop bit and U1ARX, U1ATX are used
  //U1MODE = 0x8000;     // 1-stop bit and U1RX, U1TX are used
  U1STAbits.UTXEN = 1;   // Enable the UART transmiter
  U1STAbits.UTXISEL = 0; // Interrupt generated when any character
                         // transferred to transmit register
  IEC0bits.U1TXIE = 1;   // Enable the Interrupt for the Transmiter
  IFS0bits.U1TXIF = 0;   // Clear transmiter Interrupt flag to transmit
  IEC0bits.U1RXIE = 1;   // Enable the Interrupt for the Receiver
  IFS0bits.U1RXIF = 0;   // Clear the transmiter Interrupt flag to receive
  U1BRG = 0x0003;        // Baud_rate 115200
  // U1BRG = 0x0007;     // Baud_rate 57600
}

void AD_set()  // ***** A to D (A/D) Settings
{
  ADPCFG = 0x0038; // Configure pins AN(3-5)/RB(3-5) into Digital
                   // I/O mode AN(0-2) pins are in Analog mode
  ADCON1 = 0x01E0; // SSRC bit = 111 (auto convert) implies internal
                   // counter ends sampling and starts converting.
  ADCHS = 0x0000;  // Connect RB0/AN0 as CH0 input.
  ADCSSL = 0;
  ADCON3 = 0x0F00; // Sample time = 15Tad, Tad = internal Tcy/2
  ADCON2 = 0x0004; // Interrupt after every 2 samples
}

// ***** UART Transmit ISR
void __attribute__((interrupt, no_auto_psv)) _U1TXInterrupt(void)
{
  IFS0bits.U1TXIF = 0;  // clear TX interrupt flag
}

// ***** UART Receive ISR
```

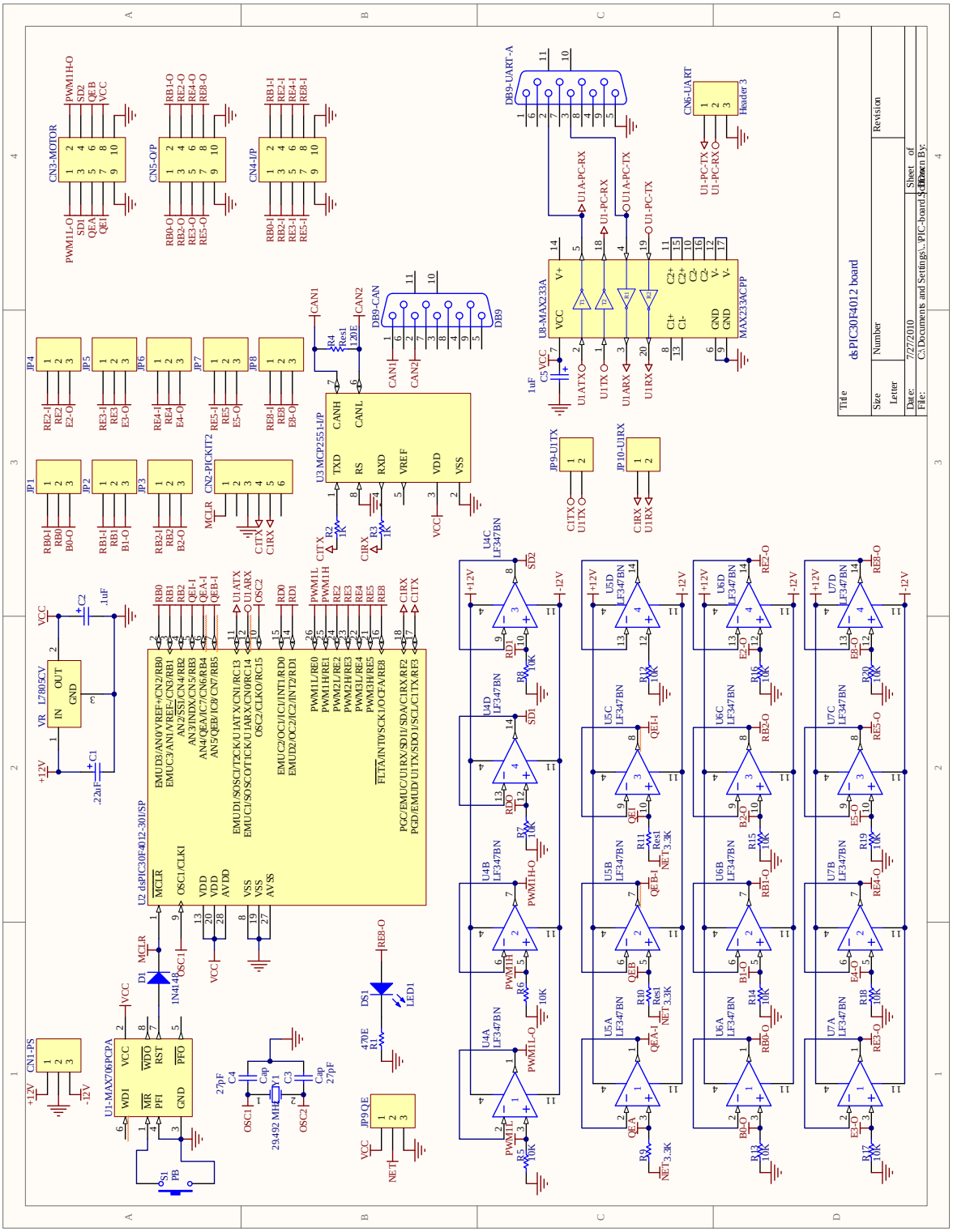
```
void __attribute__((interrupt, no_auto_psv)) _U1RXInterrupt(void)
{
  IFS0bits.U1RXIF = 0; //clear receive interrupt flag
  /*
     while (U1STAbits.URXDA)
     {
      speedset=U1RXREG;
     }
  */
  if(U1STAbits.OERR == 1)
  {
   U1STAbits.OERR = 0; // Clear Overrun Error to receive data
  }
}
```

1.8.3 readplot.m

```
% readplot.m: Uses GNU Octave's or MATLAB's dlmread function to
% read the contents of the file testdata.txt into a vector. This
% file belongs to the lab manual for EE380 (control lab).
%
% The file testdata.txt is generated as follows. The program
% terminal.exe writes the information that it receives from
% dsPIC30F4012 to the file testdata.txt. The program in
% dsPIC30F4012 sends this information as tab seperated ASCII
% values.
%
% We have tested that this m-file executes nicely in GNU Octave
% version 3.2.4 that comes packaged for Windows in
%
% http://sourceforge.net/projects/octave/files/
% Octave_Windows%20-%20MinGW/
% Octave%203.2.4%20for%20Windows%20MinGW32%20Installer/
% Octave-3.2.4_i686-pc-mingw32_gcc-4.4.0_setup.exe/download
%
% and MATLAB 7.7.0471 (R2008b) that we have in our CC. On MATLAB
% dlmread('testdata.txt') seems to be returning the last item in
% the vector as 0 even though it may be blank. GNU Octave does
% not have this problem.
%
% The plots are generated nicely in MATLAB and the Linux version
% of GNU Octave.  The plotting program (most likely GNUPlot) in
% the windows version of GNU Octave does not seem to be properly
% integrated into GNU Octave. So, we have trouble displaying the
% results of plot on the screen. As a work around, we have used
% the command
%
%   print -djpg plot.jpg
%
% to print the plots to jpeg files.
```

```
%
% We found that this problem has been reported at
%
% http://octave.1599824.n4.nabble.com/
% Gnuplot-freezes-in-Win7-3-2-4-td2279218.html#a2279218
%
% and a work-around has been suggested there and at
%
% http://old.nabble.com/
% Re:-Octave-3.2.4-mingw32-available-p28053703.html
%
% Using this work-around does remove this problem.
%
% PRECONDITIONS: readplot.m and testdata.txt need to be in the
% same folder. All data in testdata.txt is tab-separated and in a
% single row, and no spaces lead the first item of the data.
%
% Date created on: September 12, 2010.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all, close all, clc

x = dlmread('terminal.log');

% Determine the number of rows and columns of x.
% If all went well, the number of rows will be equal to 1.
[rows,cols] = size(x);

% Truncate x so that x has an even number of columns.
if cols/2 > floor(cols/2)
  x = x(:,1:cols-1);
  cols = cols-1;
end

% Extract columns number 1, 3, 5, ... into a vector w,
% and columns number 2, 4, 6, ... into a vector u.
w = x(1,1:2:cols-1); % This is the vector of speeds
u = x(1,2:2:cols)/100;   % This is the vector of voltages.

% Calculate times at which to plot speed and voltage.
T = 0.002; % sampling time
t = 0:T:T*(cols/2-1);

% Plot the speeds and the voltages with respect to time.
subplot(2,1,1); plot(t,w); grid;
title('Speed of the motor shaft in (rad/s)');
subplot(2,1,2); plot(t,u); grid(gca,'minor');
title('Voltage applied to motor in (V)');

print -djpg plots.jpg
```

1.9 Schematic of the dsPIC30F4012 board



Chapter 2

Experiment 1: PMDC motor modeling, identification, speed control

2.1 Goals

1. Develop a physics-based model for a PMDC motor.
2. For the PMDC motor develop a model based on system identification using open-loop (OL) step response.
3. Design a speed controller for the physics-based model using Bode plot-based loop-shaping techniques. Simulate this controller.
4. Redesign the speed controller for the identified model, simulate this controller, and implement it practically. Compare results.

The block diagram of the control system is shown in Figure 1.1.

Section 2.2 lists the steps that help achieve the above-stated goals. Sections 2.3 onwards fill in the technical background needed to execute these steps.

2.2 Exercises

2.2.1 To do at home

Q1 Verify equations (2.2) and (2.3) using Figure 2.2, and determine the numerical values using Table 1.1.

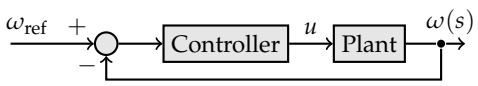


Figure 2.1: CL system to be simulated.

Q2 Design using Bode plot-based loop-shaping techniques a controller of the *minimum order possible* to control the speed of the given motor for the following time domain specifications: $e_{ss} \leq 2\%$, $t_s \approx 0.5$ s (you have upto 5% tolerance on t_s), $M_p \leq 20\%$.

A 5-cycle semilog graph paper is provided at the end of this manual.

Q3 Simulate the continuous-time controller designed above using GNU Octave. That is, simulate the CL system of Figure 2.1. You can use such GNU Octave functions as `series`, `feedback`, `cloop`, `conv`, etc. For example,

```
step(feedback(series(tf(1,[1,1]),tf(1,[2,1])),tf(1,1)));
```

If the closed-loop system performance in simulation is not as desired, you may need to redesign your controller.

Q4 Discretize the continuous-time controller with the sampling period T_s. See Section 2.5.

Q5 With the discretized version, perform a simulation of digital control of the continuous-time plant using the m-file `easysim.m` provided.

Plot your results as two subplots with ω versus t in the upper subplot and u versus t in the lower subplot.

Do you think that your digitally-controlled closed-loop system will be stable in practice? Will it provide in practice the same performance as did the continous-time version in simulation?

Q6 Write the digital controller in C.

What do you think is the difference between Q3 and Q5?

2.2.2 To do in lab

Q7 Write a code to apply a step voltage to the motor. Run the motor in OL.

Q8 Identify the values of K_m and τ_m using the OL step response. See Section 2.4.

Q9 For the identified model, redesign your controller using loop-shaping on the graph paper you used in your homework.

Q10 With the discretized version of the above-*redesigned* controller, perform a simulation of digital control of the continuous-time *identified* plant using the m-file `easysim.m`.

Plot your results as two subplots with ω versus t in the upper subplot and u versus t in the lower subplot, and show this plot to the instructor.

Q10 dropped for 2013

Q11 Program the lab-designed digital controller and run the setup.

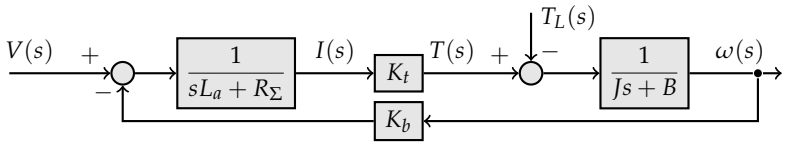


Figure 2.2: Block diagram of a PMDC motor. R_Σ includes the armature resistance R_a and other resistances as explained in Section 1.6.

Plot your results as two subplots with ω versus t in the upper subplot and u versus t in the lower subplot, and show this plot to the instructor.

Q12 Compare the performance of the controller designed for the identified model in simulation and in experiment. Fill the following table:

	M_p [%]	e_{ss}	t_s [s]	Sketch of response ω vs. t and u vs. t
Simulation				
Experiment				

Ideally, this comparison is done by plotting the two plots of ω versus t on one subplot and the two plots of u versus t on the other subplot.

Q13 Conclusions: Is the physics-based model a good match to the plant? If not, what do you think we have ignored that has lead to the difference?

2.3 Physics-based model of the DC motor unit

Since the inductance L_a is very small[1], by neglecting L_a, from Figure 2.2, the transfer function (TF) from the input voltage $V(t)$ to the speed $\omega(t)$ of the motor shaft is

$$\frac{\omega(s)}{V(s)} = \frac{K_m}{\tau_m s + 1} \tag{2.1}$$

with

$$K_m = \frac{K_T}{R_\Sigma B + K_T K_b}, \tag{2.2}$$

$$\tau_m = \frac{R_\Sigma J}{R_\Sigma B + K_T K_b} \tag{2.3}$$

Here, K_m is the motor gain constant in rad/s/V, τ_m is the motor time constant in seconds, K_T is the torque constant in $\mathrm{N \cdot m/A}$, R_Σ is the resistance in the armature path in ohms as explained in Section 1.6, B is the viscous-friction coefficient of the motor rotor with attached mechanical load in $\mathrm{N \cdot m/(rad/sec)}$,

[1]When is an inductor considered small? When the time constant due to this inductor is negligible in comparison to the remaining time constants in the TF of the system, this inductor is considered neglibly small. It can be verified using the data provided in and near Table 1.1 that for this setup, $J/B \gg L_a/R_\Sigma$

J is the moment of inertia of the motor rotor with attached mechanical load $\text{kg} \cdot \text{m}^2$, K_b is the back emf constant in V/(rad/sec).

2.4 System identification

As we know that the motor transfer function is theoretically of first order as in Equation (2.1), we can identify K_m and τ_m from the V-to-ω step response as follows:

Step 1 Apply a step input of magnitude R volts from the microcontroller to the motor.

This is achieved by generating a number r within the microcontroller that is proportional to R volts. The microcontroller converts r into a suitable duty ratio that is then applied to the H-bridge. The H-bridge will then — on the average within each sampling interval — apply a DC voltage of R volts to the motor.

Step 2 Plot the data of motor speed versus time. This data comes from the encoder to the microcontroller that, in turn, sends to the PC via UART. See Section 1.7.2.

Step 3 If the armature voltage-to-speed TF is indeed first order, and of the form $K/(\tau s + 1)$, then the speed will be the following function of time:

$$\omega(t) = RK(1 - e^{-t/\tau}) \tag{2.4}$$

Therefore, τ and K can be obtained from the plot of ω versus t as

- τ is the time where the speed is $0.6321\omega(\infty)$
- $RK = \omega(\infty)$.

To check that the response is indeed first order, we can plot ω generated by Equation (2.4) on the plot of Step 2, and see how closely the two plots match. We may even tweak τ and K to find the best match between the two plots.

2.5 Discretized version of the controller

The controller we design will be in the form of a TF. To program this controller equation in the microcontroller, we need to recognize that the microcontroller controls the motor by sampling the motor states periodically with a sampling period of T_s seconds, and that this sampling period may affect the form of the controller equation in the microcontroller.

In particular, we "discretize" the controller equation. That is, we convert the equation from continuous-time form to a discrete-time form. We perform this discretization through the following steps: convert the controller into a state-space equation, discretize this state-space equation, and use this discretized equation for a time-domain recursion. We saw this technique in EE250. Here we recap this technique.

2.5.1 Conversion from transfer function to state-space

Consider the following TF

$$G(s) = \frac{Y(s)}{U(s)} = \frac{a_1 s + a_0}{s^2 + b_1 s + b_0}$$

We wish to find a state space model whose input is $u(t)$ and output is $y(t)$, and whose TF is $G(s)$. The following is one way to accomplish this goal:

1. Introduce an intermediate variable $X(s)$ thus:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{Y(s)}{X(s)} \frac{X(s)}{U(s)}$$

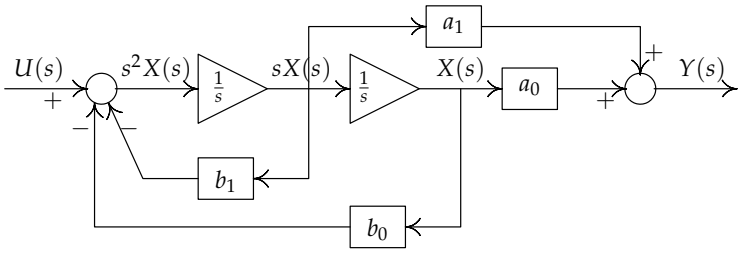
2. Let

$$\frac{X(s)}{U(s)} = \frac{1}{s^2 + b_1 s + b_0} \quad \text{and} \quad \frac{Y(s)}{X(s)} = a_1 s + a_0$$

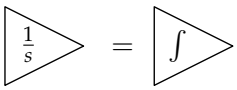
3. From the last step above, we can write the following:

$$s^2 X(s) = U(s) - b_1 s X(s) - b_0 X(s)$$
$$Y(s) = a_1 s X(s) + a_0 X(s)$$

4. Using this last step, we can construct the following simulation diagram:

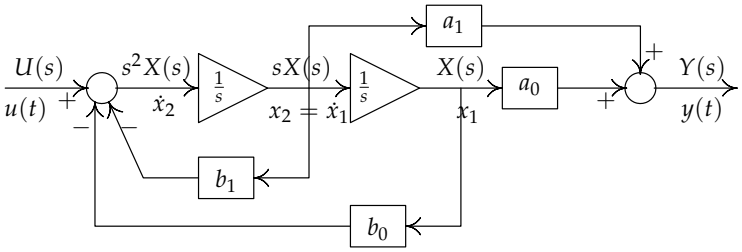


Note that here, the block containing the $1/s$ represents integration operation. Books show either of the following two equivalent blocks:



Strictly speaking, the $1/s$ is used s-domain block diagrams, while the $\int$ is used in time-domain block diagrams.

5. In the block diagram, write the time-domain quantities $u(t)$ and $y(t)$ as shown below, and assign a state variable to the output of each integrator.



For example, we start assigning at the output of the right-most integrator, and go left.

6. Write the differential equations based on the time-domain quantities in the simulation diagram:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -b_0 x_1 - b_1 x_2 + u \\ y &= a_0 x_1 + a_1 x_2 \end{aligned}$$

This gives us the following state-space (SS) model:

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ -b_0 & -b_1 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \quad \text{with } x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{2.5}$$

An easy way is to use `tf2ss` in GNU Octave or Matlab

2.5.2 Discretization of the state-space equation

Given the state-space equation $\dot{x} = Ax + Bu$, use Euler's approximation to write $\dot{x}$ as $\dot{x} \approx \frac{x(t+\Delta t)-x(t)}{\Delta t}$. Then, write this state-space equation as $x(t+\Delta t) \approx (A\Delta t + I)x(t) + B\Delta t u(t)$. This approximation works well if Δt is sufficiently small. As a rule of thumb, Δt should be at most one-tenth or one-twentieth of the smallest time constant of A.

2.5.3 Time-domain recursion

The equation $x(t+\Delta t) \approx (A\Delta t + I)x(t) + B\Delta t u(t)$ gives the following formula for recursion, with $T_s = \Delta t$:

$$x(k+1) = (AT_s + I)x(k) + BT_s u(k)$$

For example, Equation (2.5), on discretization, has the following form that is ready for recursion:

$$\begin{aligned} x_1(k+1) &= x_1(k) + T_s x_2(k) \\ x_2(k+1) &= -b_0 T_s x_1(k) + (1 - b_1 T_s)x_2(k) + T_s u(k) \end{aligned}$$

In our simulations, we may use $T_s = 0.002$ s

2.6 Simulation

The discrete-time controller will work in practice on a continuous-time plant. However, the continuous-time plant is not available at home. Therefore, we

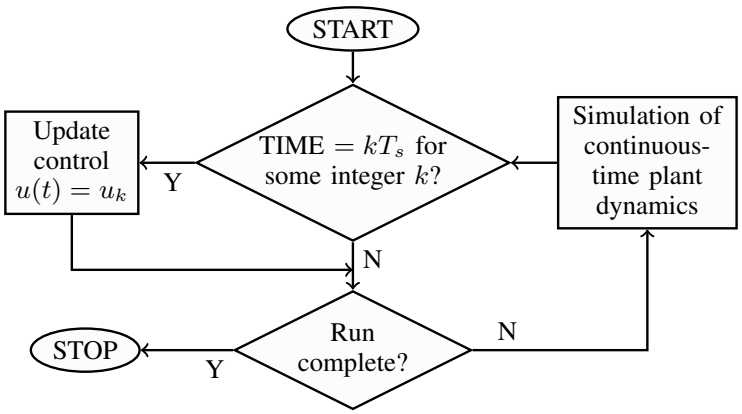


Figure 2.3: Scheme for simulation of digital control of continuous-time system, adapted from [3]. The control input $u(t)$ is updated at each time kT_s, and then held until time $(k+1)T_s$.

instead test the digital control of the continuous-time plant in simulation. The simulation scheme is shown in Figure 2.3 and is implemented in the m-file `easysim.m`. which runs on MATLAB and GNU Octave.

2.7 Program listings

2.7.1 easysim.m

```
% easysim.m: Simulates digital control of continuous-time system
% without needing Matlab's Control System Toolbox. Uses only
% Euler's approximation. Runs in Matlab and GNU Octave.
%
% PRECONDITIONS: Tp < Tc/10; Tc < Tmin/10. Here, Tmin is smallest
% time constant of CL system. User needs to convert plant and
% controller TFs to SS.
%------------------------------------------------------------
clc; clear all; close all;
% ------------------- Declarations ---------------------------
%
% Plant transfer function K/(s+w)
K = 100; w = 1;
% State space model of plant is
%
% xpdot = -w*xp + up;
% yp = K*xp;
%
% The suffix ``p'' represents plant.

% Controller will sample plant states every Tc seconds
Tc = 0.01;
```

```
% We will control the plant for tcfin seconds
tcfin = 5;

% Step size Tp used for numerical integration of plant
% differential equation using Euler's approximation.
Tp = 0.00001;
% We will numerically integrate plant differential
% equation for Tc seconds.

% Controller TF is Cs = (Kp*s+Ki)/s
Kp = 0.11; Ki = 1.737;
% State-space model of controller is
%
% xcdot = uc;
% yc = Ki*xc + Kp*uc;
%
% The suffix ``c'' represents controller.
%-------------------------------------------------------------
%-- Simulate continuous-time plant discrete-time controller --

sd = 100;    % Desired motor speed in rad/sec.
sa(1) = 0;   % Initial actual speed (sa = yp).
xc(1) = 0;   % Initial state of controller.
yc(1) = 0;   % Intial output of controller.
xp(1) = 0;   % Initial state of plant.
for k = 1:tcfin/Tc
  uc(k) = sd - sa(k);
  xc(k+1) = uc(k)*Tc + xc(k);
  yc(k) = Kp*uc(k) + Ki*xc(k);
  % Hold last sample of controller output:
  up = yc(k);
  % Numerically integrate plant equation holding the last
  % controller output as the input to the plant.
  for i = 1:Tc/Tp-1
    xp = (1-Tp*w)*xp + Tp*up;
    % This is the equation
    % xp(k+1) = (1-Tp*w)*xp(k) + Tp*up;
  end
  yp(k) = K*xp;
  sa(k+1) = yp(k);
end

t = (0:tcfin/Tc)*Tc;
plot(t,sa); grid(gca,'minor');
print -depsc Tc0-0001.eps
```

Chapter 3

Experiment 2: Speed of PMDC motor tracks reference sinusoid

3.1 Goals

Recognition of dead zone. Identification of parameters of mathematical model of a PMDC motor using least squares estimation (LSE). Design and implementation of speed controller for the identified model using loop-shaping techniques. Speed tracks a reference sinusoid.

3.2 Questions

3.2.1 To do at home

Q1 Write down the identified mathematical model you used in Experiment 1.

Q2 In the lab, we will apply a sinusoidal voltage from a function generator (FG) to the dsPIC microcontroller's analog input. We will want the motor's speed to track this sinusoidal input.

Design using loop-shaping, a controller of first order such that the closed-loop system will track sinusoids of frequencies upto 7 Hz with $e_{ss} \leq 2\%$ (in magnitude). For the settling time (defined as "time to enter the x% tube with the intention of remaining in it") do the best you can achieve, given the other specifications, and given that the imperfections of the plant are what they are. Hint: See EE250 lecture notes for a solution to this problem.

Q4 Discretize the continuous-time controller using Euler's approximation. Use `tf2ss`.

Q5 With the discretized version, perform a simulation of digital control of the continuous-time plant using the m-file `simsine.m`. Apply reference sinusoids of magnitude = 150 rad/s, and about 4 frequencies (in Hz): 1, 3, 5, 7. Modify `simsine.m` to suit your purpose. Populate the following table.

Frequency of reference sinusoid of amplitude 150 rad/s [Hz]	1	3	5	7
Amplitude of rotor speed ω in closed loop [rad/s]				
Magnitude of control u [V]				

If the desired performance is not achieved, then repeat Q2 onwards. Else, proceed to Q6.

In the lab, observe the frequencies up to which tracking happens well.

Q6 Write the digital controller in C.

Q7 SYSTEM IDENTIFICATION USING LSE: Supply various values to K, a, b in the file `sysid.m`, execute this file in GNU Octave, and compare the resulting values of K, a, b with the supplied values. Do you think that `sysid.m` is doing a good job of estimating the supplied values?

Q8 Assume that the plant TF obtained in Experiment 1 is $32.286/(0.052s + 1)$.

A voltage waveform is applied to the open-loop system from a function generator. Three sets of $u - \omega$ data are obtained into files named `tri4fg5.log`, `tri8fg5.log`, and `rect4fg5.log`. These data correspond respectively to trianglular waveform of $u \approx 4$ V amplitude, triangular waveform of $u \approx 8$ V amplitude, and rectangular waveform of $u \approx 4$ V amplitude.

To see the effect of the deadzone, plot the contents of each of the `.log` files using `readplot.m`, and sketch your results below.

ω vs. t and u vs. t from `tri4fg5.log`	ω vs. t and u vs. t from `tri8fg5.log`	ω vs. t and u vs. t from `rect4fg5.log`

Then, use the attached file `readSID.m` that is an amalgam of `readplot.m` and `sysid.m` to populate the following table.

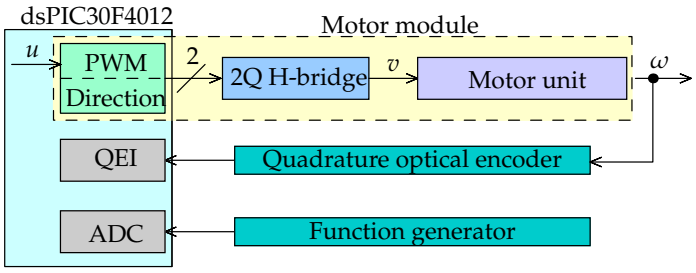


Figure 3.1: Block diagram of the experimental setup as applicable to the task of least squares estimation. The signal u is generated either by a program or is the output of the ADC. The motor unit comprises the PMDC motor with gear.

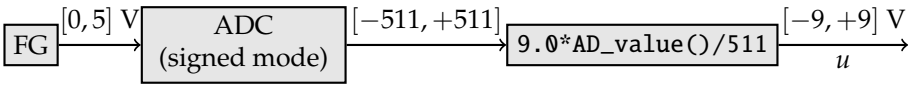


Figure 3.2: how the output of the FG is converted into a u of desired value.

Type of TF	TF	Parameters of step response	
		$\omega(\infty)$ [rad/s]	Sketch of step responses (all in one) (unfiltered ones)
TF from Exp-t 1			
TF from triangle of 4 V amplitude			
TF from triangle of 8 V amplitude			
TF from rectangle of 4 V amplitude			

3.2.2 To do in lab

Q9 We will apply a triangular voltage waveform to the armature of the PMDC motor. While a code can be inserted into `main-prog.c` to generate this waveform internally inside the μC, we will instead input the waveform from a function generator as shown in Figure 3.1.

Insert the code `u = 9.0*AD_value()/511;` into the section devoted to the controller code in `main-prog.c`[1]. Configure the FG to output a triangular waveform with an amplitude of 2.5 V, offset of 2.5 V, and frequency of 1 Hz. To configure the 2.5 V offset, you can use the fact that at this offset the pulley turns as much in the clockwise direction as in the anticlockwise direction, while if the offset is not perfect the pulley has a net creep in one direction. *This configuration needs to remain undisturbed throughout the experiment irre-*

[1]Note that the code `u = 9/511*AD_value();` does not give the same results.

spective of the waveform used. Figure 3.2 shows how this configuration helps obtain the desired u.

Run the plant in open-loop mode. Record the input-output data (that is, u, ω) into a file named `expt2-q9.log`. Our FG outputs by default a waveform of about 4 V peak-to-peak value, though the display of the FG says 5 V. You will use the data in `expt2-q9.log` to identify the plant parameters.

Q10 Identify the plant parameters using least squares estimation.

In the line `[y3,t3] = step(tf(31.42/0.07,[1,1/0.07]));` of `readSID.m`, use the plant parameters that you obtained in Q8 of Experiment 1. Then use `readSID.m` to populate the following table, and note which step response is closest to the step response of the TF from Q8 of Experiment 1.

Type of TF	TF	Parameters of step response	
		$\omega(\infty)$ [rad/s]	Sketch of step responses
TF from Exp-t 1			
TF from triangle without filter in `readSID.m`			
TF from triangle with filter in `readSID.m`			

Q11 Approximately fit a first order TF to the unit step response of the TF corresponding to the triangle (8 V ampl, with filter in `readSID.m`) in Q10, similar to how you did in Q8 of Experiment 1.

Q12 With the first order TF in Q11, redesign your controller using loop-shaping for the motor to track sinusoids upto 7 Hz with $e_{ss} \leq 10\%$. *Hint: You do not need a semilog graph paper here.*

Q12a With the discretized version of the controller from Q11, perform a simulation of digital control of the continuous-time plant as you did in Q5.

Frequency of sinusoid of magnitude 10 rad/s [Hz]	0.5	1
Amplitude of output ω of CL system [rad/s] (in simulation)		
Amplitude of control u [V] (in simulation)		

Q12a dropped for 2013. In real world controller design and implementation, this simulation is done to estimate the control effort needed when the controller is deployed on the setup.

Q13 Program the discretized version of the controller from Q10 into the dsPIC, run the setup, if necessary, very slightly adjust the offset knob of the FG so that there is no net creep of the pulley in any one direction, and populate the following table.

In `main-prog.c`, you will need to comment out the part `u =`

`9.0/511*AD_value();`, write your controller in the appropriate place, and uncomment the two lines

```
R = AD_value(); // In signed mode, ADC maps [0,5] V to [-511,+511].
R = 100.0*R/511; // R = 100*sin(w_in*t) rad/sec; 100 is max speed.
```

that correspond to this experiment.

Frequency of sinusoid of magnitude 10 rad/s [Hz]	0.5	1
Magnitude of output ω of CL system [rad/s] (in experiment)		
Magnitude of control u [V] (in experiment)		

Q14 Conclusions: What is the largest frequency sinusoid that this CL system is able to track? What is limiting this frequency?

3.3 Dead zone in the V_m versus u characteristic

The dead zone in the V_m versus u characteristic of Figure 1.7 does not matter when we control the PMDC motor for its speed to track a step as the speed is needed to be at a large value and u quickly becomes large enough, without dwelling in the dead zone.

In the position control of the PMDC motor (Chapter 9), however, u becomes small when the desired position is reached. If u becomes smaller than 2 volts, then V_m will be zero as shown in Figure 1.7, making the position control system unresponsive.

Similar to the case of position control, when the speed wishes to track a sinusoid, the reference speed, and therefore u, dips to near-zero values before gradually going to larger values. In this case, the dead zone can be experienced by using small input voltages, for example, $u = 3 \sin \omega t$.

To help the motor to remain responsive when the applied armature voltage reaches near zero values, we could either

1. use an integral component in the controller, or
2. add the following code in `main-prog.c` before or after the if condition used for limiting the duty ratio.
   `if(u<0&&u>-2) u = u - 2; else if(u>0&&u<2) u = u + 2;`

Dead zone compensation is addressed in another experiment (Chapter 5).

3.4 System identification

The plant TF (Equation 2.1) has two parameters, K_m and τ_m, that need to be identified. We use from [4, pages 503–505] a technique to obtain the best estimate of the parameters of a transfer function (TF) in the least squares sense.

Consider the second order TF

$$\frac{Y(s)}{U(s)} = \frac{K}{s^2 + as + b}. \tag{3.1}$$

Using the bilinear transformation $s = \frac{2}{T_s}(\frac{z-1}{z+1})$ with sampling period T_s on (3.1) gives

$$\frac{Y(z)}{U(z)} = \frac{(z^2 + 2z + 1)K}{\frac{4}{T_s^2}(z^2 - 2z + 1) + \frac{2}{T_s}(z^2 - 1)a + (z^2 + 2z + 1)b}.$$

The inverse z-transform with $n = 2, 3, \ldots, N$ and $k = n$ gives the discrete-time equation

$$\gamma_n = \phi_n^T \sigma$$

where

$$\gamma_n = \frac{4}{T_s^2}\left(y(k) - 2y(k-1) + y(k-2)\right),$$

$$\phi_n = \begin{pmatrix} u(k) + 2u(k-1) + u(k-2) \\ -\frac{2}{T_s}(y(k) - y(k-2)) \\ -(y(k) + 2y(k-1) + y(k-2)) \end{pmatrix}, \tag{3.2}$$

and $\sigma = \begin{pmatrix} K & a & b \end{pmatrix}^T$. σ is estimated using the data set of observed outputs $y(0), y(1), \ldots, y(N)$, and applied inputs $u(0), u(1), \ldots, u(N)$. The best estimate in the least squares sense $\hat{\sigma}$ of σ is $\hat{\sigma} = (\Phi^T\Phi)^{-1}\Phi^T Y$, where Y and Φ are

$$Y = \begin{bmatrix} \gamma_n & \gamma_{n+1} & \ldots & \gamma_N \end{bmatrix}^T, \quad \Phi = \begin{bmatrix} \phi_n & \phi_{n+1} & \ldots & \phi_N \end{bmatrix}^T.$$

The m-file `sysid.m` contains sample code for system identification. In `sysid.m` the input $u(k)$ is applied to an example plant; the output $y(k)$ is assumed to contain noise. Identification using these input and output values results in values of the system parameters that are very close to those of the plant.

3.5 Program listings

3.5.1 sysid.m

```
% sysid.m: Implements least squares system identification. The
% method is taken from Gene F. Franklin, J. David Powell, and
% Michael L. Workman. Digital Control of Dynamic Systems.
% Addison-Wesley, 3rd edition, 1998, pages 503 -- 505.
%
% Uses only Euler's approximation. User needs to convert plant TF
% to SS. Works in GNU Octave as well as MATLAB.
%
% PRECONDITIONS: Tc < Tmin/10. Here, Tmin is the smallest time
% constant of the CL system, and Tc is the period at which the uC
```

```
% samples the plant states. In the lab manual Tc is denoted Ts.
%---------------------------------------------------------------

clc; clear all; close all;

% ------------ Begin declarations -------------------------------

% Plant transfer function K/(s^2 + a*s + b)
% Parameter vector X = [K a b]'

% K=487e2; a=14.7+1e2; b=14.7e2;
K = 100; a = 5; b = 10;
% State space model of plant is
%
% x1pdot = x2p;
% x2pdot = -b*x1p -a*x2p + up;
% yp = K*x1p;
%
% The suffix ``p'' represents plant, and the suffix ``c''
% represents controller.

% The plant states are sampled every Tc seconds
Tc = 0.01;
% We control the plant in open-loop for tcfin seconds
tcfin = 5;

%------ Declarations complete ---- Start simulation -------------

% Generate triangular control input
t = (1:tcfin/Tc)*Tc;
[Rt Ct] = size(t);
uc(1) = 0;
sgn = 1;
for i = 2:Ct
  uc(i) = uc(i-1) + sgn * 0.5;
  if(uc(i)> 9.0)
    sgn = -1;
  elseif( uc(i) < -9 )
    sgn = 1;
  end
end

% Initialize
yp(1) = 0;   % Initial actual speed.
x1p(1) = 0; x2p(1) = 0;  % Initial state of the plant.

% Recursion
for i = 1:tcfin/Tc
 x1p(i+1) = x1p(i) + x2p(i)*Tc;
 x2p(i+1) = -b*Tc*x1p(i) + (1-a*Tc)*x2p(i) + Tc*uc(i);
```

```
 yp(i) = K*x1p(i);
end

% Plot
plot(t,uc,t,yp); grid; legend('Control input','Actual speed');
print -depsc sysid.eps

% We now have a set of input-output data from the plant. We use
% this data to perform system identification

y = yp; u = uc;
k = 3;
for n =1:Ct-3
  Y(n,1) = (2/Tc)^2 * (y(k) - 2*y(k-1) + y(k-2));
  P(n,:) = [( u(k) + 2*u(k-1) + u(k-2)) ...
             (-2/Tc*( y(k)-y(k-2) ) ) -( y(k)+2*y(k-1)+y(k-2) )];
  k = k+1;
end

X = (P' * P)^(-1) * P' * Y           % X = [K a b]'
```

3.5.2 simsine.m

```
% simsine.m: Simulates the response to a sine reference input
% under the digital control of continuous-time system. Uses only
% Euler's approximation. Runs in GNU Octave.
%
% PRECONDITIONS: (1) User needs to convert plant and controller
% TFs to SS. (2) Tp < Ts/10; Ts < Tmin/10. Here, Tmin is the
% smallest time constant of the CL system, Tp is the step size of
% numerical integration of the plant dynamics, and Ts is the step
% size of the numerical integration of the closed-loop system
% dynamics and equals the chosen sampling interval.
%-------------------------------------------------------------------

clc; clear all; close all;

% ------------------ Declarations ------------------------------
% Plant transfer function Kp/(s/wp+1)
Kp = 40.714; wp = 14.3;
% Using tf2ss in Octave 3.2.4
[ap,bp,cp,dp] = tf2ss(Kp,[1/wp,1]);
% This line to be changed appropriately for Octave >= 3.6.0.

% State space model of plant is
%
% xpdot = ap*xp + bp*up;
%    yp = cp*xp + dp*up;
%
```

```
% The suffix ‘‘p’’ represents plant.

% Controller samples plant states every Ts seconds
Ts = 0.002;

% The plant is controlled in closed-loop for tsfin seconds
tsfin = 20;

% Step size Tp used for numerical integration of plant
% differential equation using Euler's approximation.
Tp = 0.00001;
% We numerically integrate plant differential equation for
% Ts seconds using a step size of Tp.

% Controller TF is
%
%       (s/z + 1)
%  Kc * ---------
%       (s/p + 1)

Kc = 50/40.714; z = 44; p = 14.3;

% Controller state space equation
%
% xcdot = ac*xc + bc*uc;
%    yc = cc*xc + dc*uc;
%
% Using tf2ss in Octave 3.2.4:
[ac,bc,cc,dc] = tf2ss(Kc*[1/z,1],[1/p,1]);
% This line to be changed appropriately for Octave >= 3.6.0.
%
% The suffix ‘‘c’’ represents controller.

%-----------------------------------------------------------------
% Simulate continuous-time plant discrete-time controller
%-----------------------------------------------------------------

% Specify the sampling instants
t =  (0:tsfin/Ts)*Ts;

% Desired motor speeds in rad/sec at sampling instants
sd = 150*sin(2*pi*7*t);

% Initial conditions
sa(1) = 0; % Initial actual speed (sa = yp).
xc(1) = 0; % Initial state of controller.
yc(1) = 0; % Intial output of controller.
xp(1) = 0; % Initial state of plant.

% Recursion
```

```
for k = 1:tsfin/Ts
 uc(k) = sd(k) - sa(k);
 xc(k+1) = (1+ac*Ts)*xc(k) + bc*Ts*uc(k);
 yc(k) = cc*xc(k) + dc*uc(k);
 % Hold last sample of controller output
 up = yc(k);
 % Numerically integrate plant equation holding
 % the input as last controller output:
 for i = 1:Ts/Tp-1
  xp = (1+Tp*ap)*xp + Tp*bp*up;
  yp = cp*xp + dp*up;
 end
 sa(k+1) = yp;
end

t = (0:tsfin/Ts)*Ts;
subplot(2,1,1); plot(t,sd, t,sa); grid(gca,'minor');
legend('reference','speed');
subplot(2,1,2); plot(t(:,1:size(yc,2)),yc); grid(gca,'minor');
legend('controller output');
print -depsc Ts0-0001.eps

% Determine the amplitudes of sa and yc after a transient.
max(sa(:,0.8*tsfin/Ts:tsfin/Ts)),
max(yc(:,0.8*tsfin/Ts:tsfin/Ts))
```

3.5.3 readSID.m

```
% readSID.m: Is an amalgam of readplot.m and sysid.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all, close all, clc


% ---------------------- Step 1 -----------------------------
% ------------ This part is from readplot.m -----------------

x = dlmread('terminal.txt');

% Determine the number of rows and columns of x.
% If all went well, the number of rows will be equal to 1.
[rows,cols] = size(x);

% Truncate x so that x has an even number of columns.
if cols/2 > floor(cols/2)
  x = x(:,1:cols-1);
  cols = cols-1;
end
```

```
% Extract columns number 1, 3, 5, ... into a vector w,
% and columns number 2, 4, 6, ... into a vector u.
w = x(1,1:2:cols-1); % This is the vector of speeds
u = x(1,2:2:cols)/10;   % This is the vector of voltages.

% Calculate times at which to plot speed and voltage.
Tp = 0.002;   t = 0:Tp:Tp*(cols/2-1);


% ---------------------- Step 2 ----------------------------
% ------------- This part is from sysid.m ------------------

% We now have a set of input-output data from the plant. We
% use this data to perform system identification.

y = w; u = u;
k = 3;
for n =1:cols/2-3
  Y(n,1) = (2/Tp)^2 * (y(k) - 2*y(k-1) + y(k-2));
  P(n,:) = [( u(k) + 2*u(k-1) + u(k-2)) ...
            (-2/Tp*( y(k)-y(k-2) ) ) -( y(k)+2*y(k-1)+y(k-2) )];
  k = k+1;
end

X = (P' * P)^(-1) * P' * Y;           % X = [K a b]'

[y1,t1] = step(tf(X(1,:),[1,X(2,:),X(3,:)]));


% ---------------------- Step 3 ----------------------------
% Redo the system identification on a low-pass filtered
% version of w.

om = 25; x(1) = 0;
for k = 1:cols/2
 x(k+1) = (1-om*Tp)*x(k) + om*Tp*w(k);
end

w = x(1:cols/2); % Filtered w.

% Use data of u and filtered w to do system identification.

y = w; u = u;
k = 3;
for n =1:cols/2-3
  Y(n,1) = (2/Tp)^2 * (y(k) - 2*y(k-1) + y(k-2));
  P(n,:) = [( u(k) + 2*u(k-1) + u(k-2)) ...
            (-2/Tp*( y(k)-y(k-2) ) ) -( y(k)+2*y(k-1)+y(k-2) )];
  k = k+1;
```

```
end

X = (P' * P)^(-1) * P' * Y;           % X = [K a b]'

[y2,t2] = step(tf(X(1,:),[1,X(2,:),X(3,:)]));

[y3,t3] = step(tf(31.42/0.07,[1,1/0.07]));

plot(t1,y1,'r',t2,y2,t3,y3,'g'); grid(gca,'minor');

legend('Step response of TF using u and w', ...
'Step response of TF using u and filtered w',...
'Step response of TF identified in Exp-t 1');
```

Chapter 4

Experiment 3: Ziegler-Nichols tuning of speed controller of PMDC motor

4.1 Goals

To apply a Ziegler-Nichols tuning (ZNT) methods to tune the parameters of a PID controller of the speed of a pmdc motor.

4.2 What is controller tuning?

Figure 4.1 shows PID control of a plant.

"If a mathematical model of the plant can be derived, then it is possible to apply various design techniques for determining parameters of the controller that will meet the transient and steady-state specifications of the closed-loop (CL) system. However, if the plant is so complicated that its mathematical model cannot be easily obtained, then an analytical approach to the design of a PID controller is not possible. Then we must resort to experimental approaches the tuning of PID controllers."[5].

"The process of selecting the controller parameters to meet given performance specifications is known as controller tuning."[5].

In practice, tuning is used in the following three situations:

1. We have designed a controller and analyzed its performance on paper and using simulation tools such as Matlab and GNU Octave. We wish to deploy this controller on the actual plant. We find that this controller does not give us the kind of behavior in practice that it gave in simulation. The design has only brought us roughly near what we want. Tuning of the parameters of the controller is needed to obtain the behavior that we saw in simulation. For

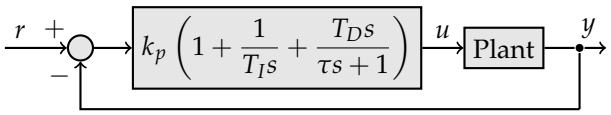


Figure 4.1: The CL system. The ZNT methods help tune k_p, T_I, T_D. The time constant τ has two purposes. Firstly, it makes implementing the derivative controller possible. Secondly, it helps create a low pass filter for high frequency noise. τ is selected to be between $0.1T_D$ and $0.2T_D$ [9, page 161].

example, "Alogrithms parameters have to be tuned to adjust to current plant, as the final plant always differs from the plant model assumed early in the design process — when the original algorithm was selected" [6].

2. The plant model may not be known; so we may not have designed a controller. We may use on-line estimation and tuning to stabilize the CL system.

3. We know that the plant belongs to a class of plants, and that for these plants a certain kind of controller will work. So, we bypass the design stage and go straight to tuning for a plant in this class. This is the situation that is addressed by the two Ziegler-Nichols tuning (ZNT) techniques.

4.3 What the two ZNT methods do

The design specifications for both the ZNT methods is quarter amplitude decay (QAD) [7]. What this means is that the tuned controller will impart the CL unit step response a second overshoot whose ratio to the first overshoot is 25% [8, page 240, Figure 4.14][1].

The ZNT rules "suggest a set of k_p, T_I, and T_D that will give a stable operation of the system. However, the resulting system may exhibit a large maximum overshoot in the step response, which is unacceptable. In such a case we need a series of fine tunings until an acceptable result is obtained. In fact, the ZNT tuning rules give an educated guess for the parameter values and provide a starting point for fine tuning, rather than giving the final settings for k_p, T_I and T_D." [5].

The first method is a time-domain method, while the second is a frequency domain method[7]. In the remainder of this section, we will describe these methods based on [5].

Though [9, page 162] only says that the second method is valid only for open-loop stable plants, we can see in the following that, as an "S" curve can be obtained for only such plants, the first method too is valid only for such plants.

4.4 First method

This method applies to plants whose step response roughly resembles an "S".

[1] The fifth edition of this book does not have a chapter on ZNT.

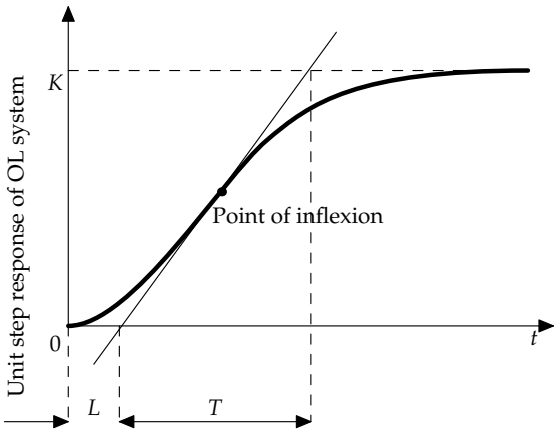


Figure 4.2: Unit step response of the plant is needed for the first ZNT method. This response is also known as the *reaction curve* of the plant. As the first ZNT method uses these curves, the first method is also known as reaction curve method. See [9, page 167] for another way of obtaining the reaction curve.

Table 4.1: ZNT rules for the First ZNT method [5, 7].

Controller	k_p	T_I	T_D
P	$\frac{T}{LK}$	∞	0
PI	$\frac{0.9T}{LK}$	$\frac{L}{0.3}$	0
PID	$\frac{1.2T}{LK}$	$2L$	$0.5L$

Step 1: Obtain the unit step response of the plant in open loop (OL); see Figure 4.2.

Step 2: Determine T and L as shown, by drawing a tangent as shown to the step response through the point of inflexion. Alternatively draw a tangent to this curve of maximum slope.

Step 3: The approximate transfer function (TF) of the OL plant is

$$\frac{\omega(s)}{U(s)} = \frac{Ke^{-Ls}}{Ts+1}$$

Step 4: Tune the PID controller parameters according to Table 4.1.

4.5 Second method

This method is also called the *ultimate gain method*. This method applies to plants that exhibit sustained oscillations in CL under proportional control for

Table 4.2: ZNT rules for the Second ZNT method [5, 7].

Controller	k_p	T_I	T_D
P	$0.5k_{cr}$	∞	0
PI	$0.45k_{cr}$	$(1/1.2)P_{cr}$	0
PID	$0.6k_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

some value of $k_p > 0$.

Step 1: Form the CL system with $k_p > 0$, $T_I = \infty$, and $T_D = 0$.

Step 2: Apply a step input to the CL system and observe its response.

Step 3: With the step input on, increase the value of k_p from 0 to some value k_{cr} (called the "critical gain" or "ultimate gain") at which the CL system will exhibit sustained oscillations.

Step 4: Determine the period P_{cr} of these oscillations.

Step 5: Tune the PID controller parameters according to Table 4.2.

4.6 A modification of the plant

As the identified plant model that we determined in Experiment 1 is of first order of the form

$$G(s) = \frac{K_m}{\tau s + 1},$$

neither ZNT method is applicable to such a plant model.

At the time of preparing this write-up, we determined that the first ZNT method gave a value of k_p that made the CL system's smallest time constant to be of the same order as the sampling period 0.005 ms, and the CL system was unstable. Indeed, any CL system, on digitization of the controller, remains stable only if the sampling period is at most about one-tenth to one-twentieth of the smallest time constant of the continuous-time CL system.

An unstable CL system meant that we would not have a meaningful experiment with ZNT and DC motor control — not a happy situation.

In order to have a meaningful experiment, we prefix a transfer function $H(s)$ (called a shaping filter) that imparts dominant poles that are closer to the origin, as well as allows us to work with a problem that is amenable to the second ZNT method too. Thanks to these poles, the k_p obtained from the ZNT methods is such that the real part of the CL system's dominant poles is almost 2 orders of magnitude smaller than the sampling frequency.

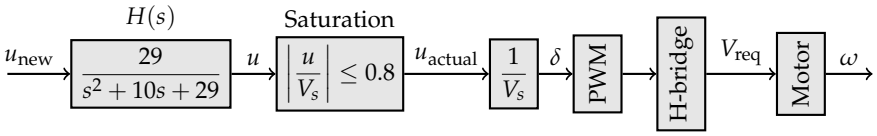


Figure 4.3: Block diagram of the plant for which we wish to tune a PID controller. Here, u_{new} is controller's output, and u_{actual} is numerical value of voltage applied to motor's armature. The duty ratio $\delta = u_{\text{actual}}/V_s$ is input to the PWM module of the dsPIC30F4012. The dsPIC outputs a PWM signal to the Solarbotics H-bridge board. The H-bridge board is fed from a dc power supply at a voltage of V_s. The H-bridge board outputs a variable-magnitude dc voltage V_{req} to the PMDC motor. The prefilter $H(s)$ is not part of the plant model that we identified in Experiment 1 or 2. We have added $H(s)$ here to make the plant model amenable to ZNT methods, as described in Section 4.6. Note that the u to ω path shown in this figure is more accurate than that shown in Figure 3.1.

4.7 Questions

4.7.1 To do at home

Q1 Write down the voltage-to-speed TF identified in Experiment 1.
Note 4.1. *This TF is the TF of the u-to-ω path in Figure 4.3.*

Q2 Prefix $H(s) = 29\,/\left(s^2+10s+29\right)$ to this TF.

Second ZNT method

Q3 Determine k_{cr} using `rlocus` of GNU Octave upto the resolution of the human eye. Simulate the CL system in GNU Octave and determine k_{cr} and P_{cr}. Fill the following table.

Value of k_{cr}	From `rlocus`	
	From simulation	
Value of P_{cr} [s]	From `rlocus`	
	From simulation	

Q4 Determine the three controllers (P, PI, PID).
Note 4.2. *Form the PID controller as shown in Figure 4.1.*

Q5 In GNU Octave, simulate the CL step response using any of `tf`, `conv`, `step`, `feedback`, `series`, etc with each of these controllers; fill the following table.

	Value of $C(s)$	t_s	e_{ss}	M_p	$\frac{2^{\text{nd}}\text{ overshoot}}{1^{\text{st}}\text{ overshoot}}$
P					
PI					
PID					

Q6 Based on the values of t_s that you have in the above table, and given that the sampling interval is 2 ms, do you think the CL system under digital control will be stable?

Q7 Repeat Q5 using `easysim.m` and $T_s = 2$ ms. Is the CL system stable?

Q8 From the above table, which controller would you select to control your plant?

Q9 Is the design specification of QAD satisfied by any of the three controllers?

Q10 Write the Euler's approximation-based descretized versions of each of the three controllers and $H(s)$ in C code. You will take these codes to the lab.

4.7.2 To do in lab: Second ZNT method

Q11 Program the discretized version of $H(s)$ in your dsPIC. Use the k_{cr} obtained in Q3. Give a step input and see the CL response. If sustained oscillations of the CL system are not seen, then tune k_{cr} until you hit a value that provides sustained oscillations. Note this value of k_{cr} and the corresponding value of P_{cr}.

Q12 With the value of k_{cr} and P_{cr} determined in Q11, form a PID controller.
Note 4.3. *Note 4.2 applies here too.*

Q13 Check using `easysim.m` that this PID controller works. Note its performance in the following table.

Type of experiment	t_s [s]	e_{ss} [%]	M_p [%]	$\frac{2^{\text{nd}} \text{ overshoot}}{1^{\text{st}} \text{ overshoot}}$
Simulation				

This question dropped for 2013

Q14 Program the digital controller from Q13 into the dsPIC and run the setup. Record the results in the following table. Plot the necessary data.

Type of experiment	t_s [s]	e_{ss} [%]	M_p [%]	$\frac{2^{\text{nd}} \text{ overshoot}}{1^{\text{st}} \text{ overshoot}}$
Practical				

Q15 Explain the differences between practice and simulation, if any.

This question dropped for 2013

Chapter 5

Experiment 4: Control of speed using armature current

5.1 Goals

To control the speed of the pmdc motor using feedback of current.

5.2 Application of this problem

The area of *sensorless speed control* (SSC) uses the measurements of the voltage V applied to the armature, and one of either armature current i or back emf E, to estimate the motor speed ω:

1. *Back emf speed control:* V is turned alternatively on and off, as in the case of our H-bridge circuit. During the on period, the motor gathers speed. During the off period, a brief period is allowed to elapse until i dies down while the rotor is *coasting*, and then the voltage is measured across the armature. This voltage is the estimate $\widehat{E}$ of the back emf. Then the estimate $\widehat{\omega}$ of ω can be obtained as $\widehat{E}/K_b$. Note that i is not measured in this method.

   Commerical controllers, such as [10], exist for SSC of dc motor using feedback of back emf. Also, see, for example, [11] for some details.

2. *Speed control using armature current:* An estimate of the back emf is obtained by measuring the armature current i and using the expression $E = V - R_\Sigma i$.

   Commercial controllers, such as [12], exist for SSC of pmdc motor using IR compensation. Also, see, for example, [13] for some details.

The advantage of SSC is that V and i can be measured using components that are fixed in the motor circuit. On the other hand, speed is measured using devices, such as encoders or tachogenerators. There are several drawbacks in using these devices such as increased volume of the motor unit and lowered

reliability as the encoders may go bad earlier than the motors. The latter drawback may be particularly significant where repair may not be possible, such as in space applications.

5.3 Background

The equations that describe a pmdc motor are

$$V = L\frac{di}{dt} + Ri + E \tag{5.1}$$

$$J\frac{d\omega}{dt} = -B\omega + T - T_L \tag{5.2}$$

Here, $T = K_T i$, $E = K_b\omega$, and V is the V_{req} of Figure 4.3. Depending on the situation, the load torque T_L may either be treated as a disturbance or as an input. In this experiment, we will treat it as a disturbance.

The resistance R needs to be seen between the terminals across which the voltage V is applied. As explained in Figure 5.1, $R = R_\Sigma$. Therefore, we have

$$V = L\frac{di}{dt} + R_\Sigma i + E$$

We see from Figure 2.2 that there are two time constants in the V to ω transfer function of the pmdc motor. These are the electrical time constant $\tau_e = L/R_\Sigma$ and the mechanical time constant $\tau_m = J/B$. As a rule of thumb, in a general pmdc motor, τ_e is about one tenth τ_m. However, in our case, the τ_e is 1.9×10^{-5} s, while τ_m is 0.44 s. That is, transients in i settle about 10^4 times as fast as transients in ω, meaning that τ_e can be ignored without any loss of accuracy.

To remove τ_e from discussion, we set $L = 0$, resulting in $V = R_\Sigma i + K_b\omega$. This equation shows that, when $\tau_e \approx 0$, ω can be calculated as $\omega = \frac{V - R_\Sigma i}{K_b}$. We can use this equation to determine $\widehat{\omega}$ and use this estimate for feedback control of ω as shown in Figure 5.2.

5.4 Questions

5.4.1 To do at home

Q1 Using the values of K_m and τ_m that you determined from the OL step response in Experiment 1, determine the values of B and R_Σ as follows.

Solve equations (2.2) and (2.3) simultaneously to determine the values of $R_\Sigma = R_H + R_m + R_{\text{sens}}$ and B. Use the values of J, K_T, K_b from Table 1.1. Do not use the value of B calculated in Section 1.6.

If you obtain $B < 0$, then you will need to find K_m and τ_m once again using the method of Section 2.4. As you cannot do this at home, use the value of B given in Section 1.6.

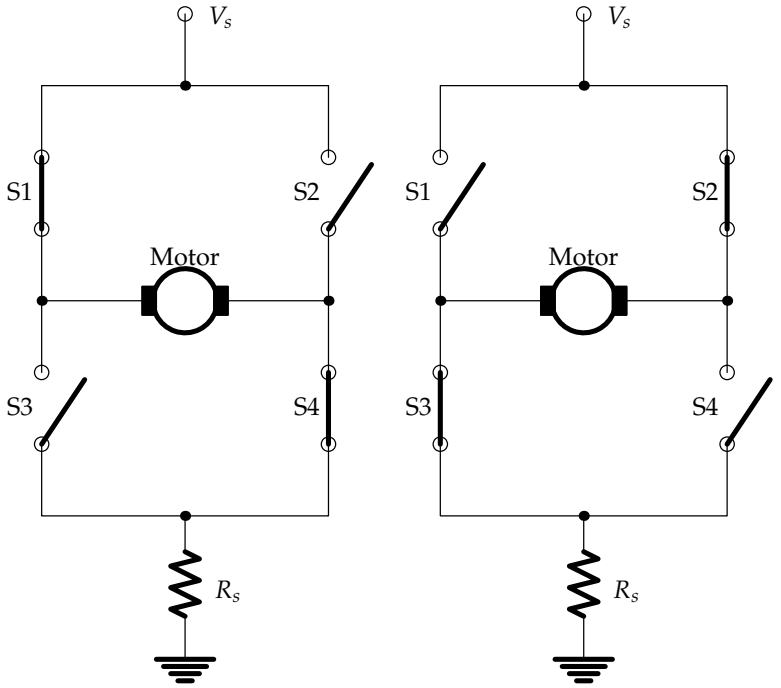


Figure 5.1: How the H-bridge is used to switch the direction of rotation of the motor shaft. With the switches S2 and S3 open, the pulse width modulated joint opening and closing of switches S1 and S4 make the motor turn in one direction with varying speed. Similarly, with the switches S1 and S4 open, and S2 and S3 closing jointly by the PWM signal the motor turns in the other direction. In either direction, the effective resistance R_Σ of the motor's armature includes the resistance R_H of the two closed switches, the resistance R_s of the sensing resistor, and the resistance R_m of the motor armature, as mentioned in Section 1.6. Also, the V_{req} of Figure 4.3 is actually the time average of the PWM voltage seen between the terminal labeled V_s and ground in the above diagram, and not the voltage immediately across the motor terminals.

Remark 1. *The* equivalent internal resistance *(R_H) of the L298 as measured by Solarbotics is as follows*[1]*:*

	IC voltage		
	9 V	*12 V*	*18.95 V*
Equivalent internal resistance (Ω)	*2.86*	*2.39*	*2.00*

However, our results of Experiment 1 give us a value of R_H that is quite different from what the above table says.

Q2 Write down the controller that you designed in Experiment 1. Show the discretized version of this controller.

Q3 Simulate your closed-loop control of motor speed in two ways under this

[1]Page 9 of `solarbotics_l298_compact_motor_driver_kit.pdf` that is available at `http://www.solarbotics.com/products/k_cmd/resources/`

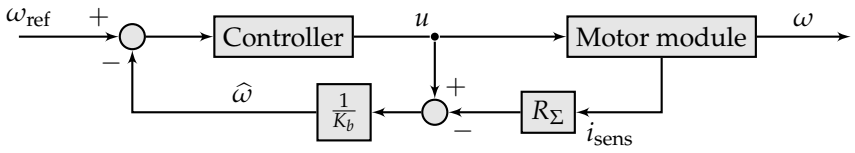


Figure 5.2: Block diagram of control of speed using feedback of armature current.

controller using a modification of `easysim.m`:

Q3.1 Using feedback of speed.

Q3.2 Using feedback of current with $\widehat{\omega} = \frac{u - R_\Sigma i}{K_b}$, and

Plot ω and $\widehat{\omega}$ on one figure.

5.4.2 To do in lab

Verify that your experimental setup is connected as appropriate for this experiment. "Appropriate" here means that the lead from the sensing resistor R_s needs to be connected to CN4 input pin 1 as shown in Figure 5.3, apart from the usual connections for speed control.

Uncommment the following parts of `main-prog.c`
`// IV = AD_value(); // Read voltage across Rs=4.7ohm.`
`// IV = 5*(511 + IV)/1022; // Convert signed to unsigned.`
and
`// Is = IV/4.7; // Convert voltage to current.`
`// IF = (1-5.0*T)*IF + 5.0*T*Is; // Low-pass filter.`

There are other parts of `main-prog.c` that you will uncomment at your discretion.

Q4 Write a code to apply a step input to the motor. Run the setup in OL mode. Identify the system parameters K_m and τ_m by using the OL step response.

Q5 If K_m and τ_m are different from those you saw in Experiment 1, calculate the values of R_Σ and B as you did in Q1.

Else, in the following, use the R_Σ determined at home.

Q6 Control the motor in the following ways:

Q6.1 Using feedback of speed (as in Experiment 1).

Q6.2 Using feedback of current with $\widehat{\omega} = (u - R_\Sigma i_{\text{sens}}) / K_b$, and

Q6.3 Using feedback of current with $\widehat{\omega} = \left(u - R_\Sigma \widehat{i}\right) / K_b$, where, $\widehat{i} \approx \frac{1}{1.8} i_{\text{sens}} - \frac{1}{30}$.

See Figure 1.7 and Section 1.5 for an explanation of this relation between i and i_{sens}.

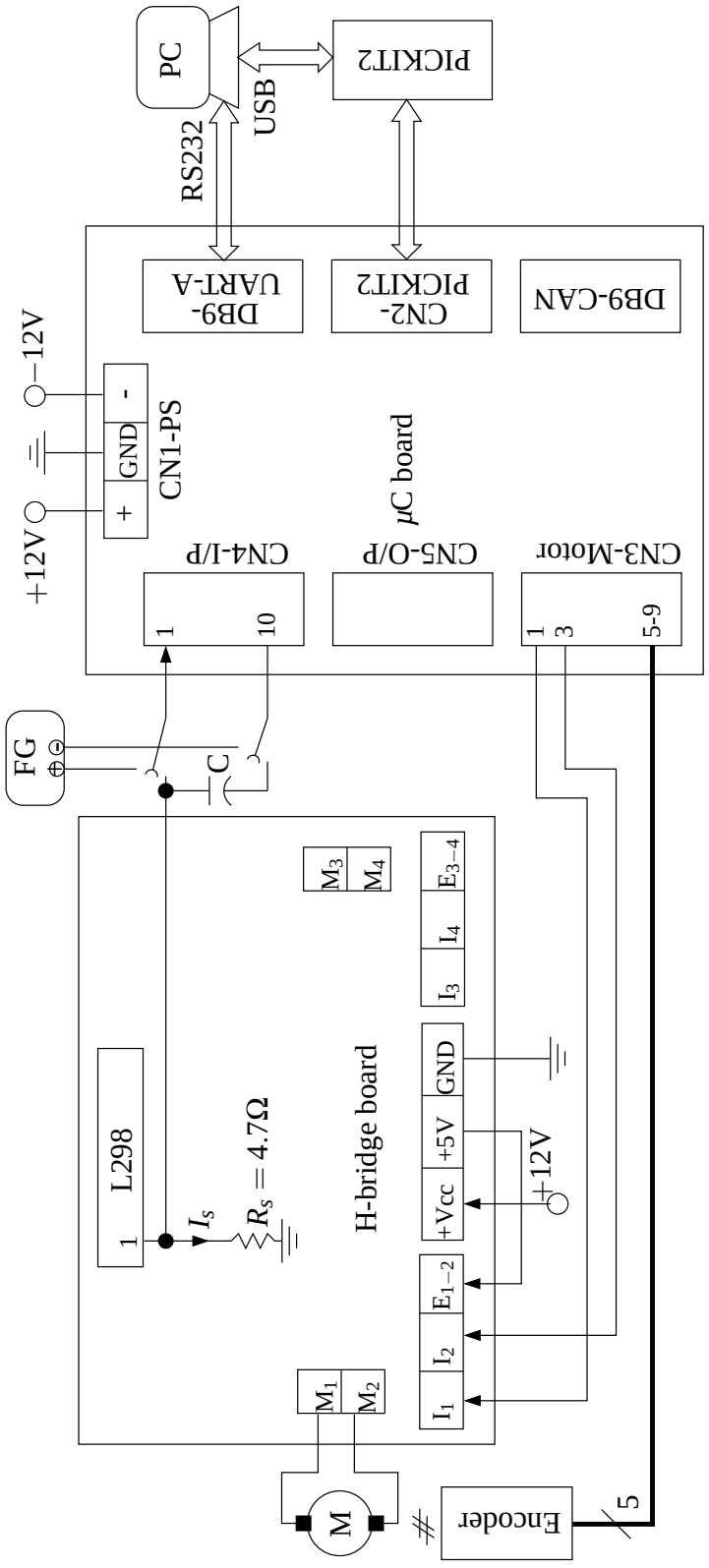


Figure 5.3: Connection diagram of the setup. In Experiment 2, a signal from the FG was input at connector CN4-I/P of the μC board. On the other hand, in the present experiment, the motor current is instead input at this connector. The capacitor C = 1000 μF.

You may need to tweak this relation between i and i_{sens} for your own setup to obtain satisfactory results. See Section 5.6 for a systematic and quick way to determine the relation between i and i_{sens}.

Q6.4 Using feedback of current with $\widehat{\omega} = \left(u - R_\Sigma \widehat{i}\right) \Big/ K_b$, where, $\widehat{i} \approx 2.5 i_{\text{sens}}$.

In each of the above cases, plot ω vs. t and $\widehat{\omega}$ vs. t as subplots on figure, and ω vs. t and u vs. t as subplots on another figure. That is, you will show us two figures for each of Q6.2 through Q6.4.

In the control of motor speed, is the feedback of armature current an adequate substitute for the feedback of motor speed?

5.5 Explanation for the C code related to currents

5.5.1 Reading the current through ADC

Here we explain the following C code.

```
IV = AD_value();  // Read voltage across Rs=4.7ohm.
IV = 5*(511 + IV)/1022; // Convert signed to unsigned.
Is = IV/4.7; // Convert voltage to current.
```

The 10-bit ADC in dsPIC30F4012 is capable of working with inputs in the range $[0, 5]$ V. There are two modes in which this ADC can work. These are *signed* and *unsigned* modes. In signed mode the ADC maps $[0, 5]$ V to the interval $[-511, +511]$. In the unsigned mode the ADC maps $[0, 5]$ V to $[0, +1023]$.

With the experiment involving tracking a sinusoid in mind, we configured the ADC to work in the signed mode. This configuration is done in `settings-prog.h` that we keep unchanged throughout our lab. When we wish to output unipolar signals from the ADC, we can either (a) reconfigure it in unsigned mode or (b) allow it to work in signed mode, but provide an offset to its ouput.We take this second approach in the above code.

5.6 Systematic method to determine i versus i_{sens}

For whatever reasons, i_{sens} in the block diagram of Figure 5.2 is not equal to i, the armature current. From Figure 1.7 that we obtained through certain trials before we developed these experiments, we estimated that

$$i_{\text{sens}} = \frac{1}{a(T_L)} i - \frac{b(T_L)}{a(T_L)}$$

Thus, we can construct an estimate $\widehat{i}$ of i using i_{sens} as

$$\widehat{i} = a(T_L) i_{\text{sens}} + b(T_L)$$

and use this estimate to construct $\widehat{\omega}$ as

$$\widehat{\omega} = \frac{V - R_\Sigma \widehat{i}}{K_b}.$$

How to determine a and b? As we have two unknowns at each value of T_L, we need two equations in these unknowns. We can proceed as follows.

Step 1 Construct $\widehat{\omega}$ as shown in Figure 5.4.

From this figure, we see that

$$\widehat{\omega}(s) = \left[U(s) - \left(\frac{I(s)}{a} - \frac{b}{as}\right) R_\Sigma\right] \frac{1}{K_b},$$

with

$$I(s) = \frac{1/R_\Sigma}{1 + \frac{K_t K_b}{R_\Sigma (Js+B)}} U(s)$$

For $u(t)$ being a step input of u_m V, the steady state value of $\widehat{\omega}$ is

$$\widehat{\omega}_{ssu_m} = \left[u_m - \frac{u_m/a}{1 + \frac{K_t K_b}{R_\Sigma B}} + \frac{bR_\Sigma}{a}\right] \frac{1}{K_b} \tag{5.3}$$

In this equation, K_t, K_b, R_Σ, B are known.

Step 2 Apply $u(t)$ of a known magnitude and measure the corresponding $\widehat{\omega}_{ssu_m}$. Then, only a and b remain as the unknowns in (5.3).

Thus, for example, for $u_m = 7$ V, we have the equation

$$a(K_b \widehat{\omega}_{ss7} - 7) = R_\Sigma b - \frac{7}{\alpha}, \text{ where } \alpha = 1 + \frac{K_t K_b}{R_\Sigma B}.$$

and for $u_m = 9$ V, we have the equation

$$a(K_b \widehat{\omega}_{ss9} - 9) = R_\Sigma b - \frac{9}{\alpha}.$$

Assuming that these two equations are linearly independent, we have

$$\begin{bmatrix} b \\ a \end{bmatrix} = \begin{bmatrix} R_\Sigma & 7 - K_b \widehat{\omega}_{ss7} \\ R_\Sigma & 9 - K_b \widehat{\omega}_{ss9} \end{bmatrix}^{-1} \begin{bmatrix} 7 \\ 9 \end{bmatrix} \times \alpha$$

Remark 2. *Ideally, we need to determine a and b for a series of values of T_L. But, in our experiment, we restrict ourselves to $T_L = 0$.*

Remark 3. *To determine $\widehat{\omega}_{ssu_m}$ see if it is not enough to observe this value from only the* `terminal.log` *file. This may save you the trouble of plotting $\widehat{\omega}$.*

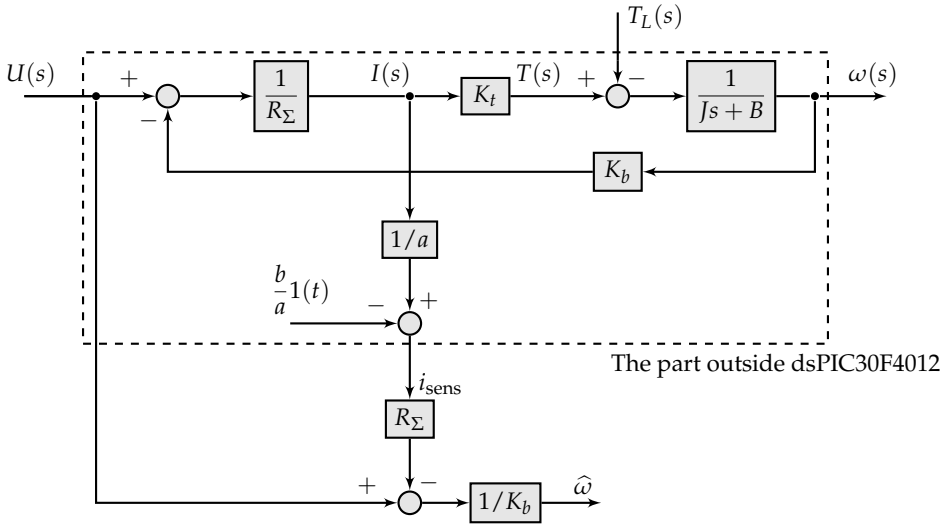


Figure 5.4: Block diagram to illustrate the procedure of Section 5.6.

5.7 Post-experiment discussion from 2011

In the experiment with $\omega_{\text{ref}} = 100$ rad/s, we found that in Q6.2 $\widehat{\omega} \approx 100$ rad/s and $\omega \approx 50$ rad/s, while in Q6.3, in the best case, $\widehat{\omega} \approx 100$ rad/s and $\omega \approx 70$ rad/s. Clearly, much improvement is needed. Prof. Avinash Joshi suggested that such a large error is most likely due to even small errors in the i — i_{sens} relationship used, or the neglected constant voltage drop across the brushes, etc. We need to pursue this line of investigation.

Thanks 1. *We thank Prof. Avinash Joshi for discussing with us the results of this experiment. From this discussion, we learned that what we are doing in this experiment goes under the name of* speed control using IR compensation*, and that IR compensation, and thereby the speed control, is indeed difficult.*

Chapter 6

Experiment 5: Control of armature current

6.1 Goals

To control the armature current of the pmdc motor at the desired value.

6.2 Application of the problem

The disturbance observer (DOB) that we see in Experiment 6 requires the armature current of the pmdc motor to be well-regulated. In the present experiment, we attempt to make the armature current well-regulated.

Apart from the application in the DOB, a current loop, when present, is the innermost loop in electric drives, the next outer one being the speed control loop, followed by the position control loop, which is also the outermost loop as shown in Figure 6.1. This three-loop structure is used also with motors other than pmdc. This structure is used for tracking a given θ_d while restricting the speed or current. When only control of speed is needed while constraining the current, the outer-most loop can be removed. See, for example, [14, page 36].

6.3 Well-regulated current

Figure 6.2 shows the block diagram of the pmdc motor under control of armature current. Figure 6.3 shows this block diagram redrawn. Figure 6.3 shows that i cannot track i_d alone, but only $i_d - i_{\text{ex}}$. The current i will be considered as well-regulated if it tracks i_d nicely; i will not track i_d nicely as long as i_{ex} dominates.

If $K_i(s)$ has poles, then $K_b/K_i(s)$ will contain zeros. Zeros represent derivative elements. Especially during transients in ω, these zeros will differentiate ω with respect to time, leading to a large value of i_{ex}, and consequently to large errors between i_d and i. Also, these zeros amplify sensor noise. Therefore, we prefer a $1/K_i(s)$ that does not have zeros. However, if we do not have a choice, then we need to use a filter on i_{sens}, which is the sensed version of i. This filter is meant to remove any noise, which the capacitor that is in parallel with R_s, may not have blocked. In our trials we found that the software filter in `main-prog.c`

```
IF = (1-5.0*T)*IF + 5.0*T*Is;  // Low-pass filter.
```

seems to be rejecting this noise adequately.

An easy way to ensure that i tracks i_d is to use for $K_i(s)$ a proportional controller with a large gain. The problem with the proportional controller is that it provides a high gain in the transient as well as steady state phases of the signals. This means that when the error between i_d and i is large, the demand on u is large, and may exceed the $9-10$ V that our H-bridge provides.

Another choice for $K_i(s)$ is a PI controller. The virtue of a PI controller is that, in the transient phase of the signals, as the frequencies are high, it offers a small gain, and therefore a small u. In the steady-state phase of the signals, as the frequencies are small, it provides a high gain to nullify the error between i_d and i.

The drawback of the PI controller is that, as $1/K_i(s)$ is a high-pass filter, $K_b/K_i(s)$ amplifies noise. However, the software filter shown above helps.

Remark 4. *Note that, in the lab, we measure i using a resistor R_s placed in the armature path as shown in Figure 5.1. However, that location of R_s tells us that i is non-negative, irrespective of the true sign of i. A scheme that gives the sign of i as well as its magnitude is shown in Figure 6.4. As our present experiment is designed such that i does not become negative, the scheme of Figure 5.1 is adequate for now.*

6.4 To do at home

Q1 Using the voltage equation (5.1) and the fundamental torque equation (5.2), and $T = K_t i$, determine the current at steady-state speed with $V = 9$ V and $T_L = 0$. Call this current i_{d1}. See the lecture slides.

The figure i_{d1} is the maximum value that we wish to specify as reference for the current control at $T_L = 0$. Any greater value of reference current at $T_L = 0$ will require the H-bridge to apply a voltage $V > 9$ V, and thereby go into voltage saturation. We wish to avoid saturation so that we may work with an approximately linear plant.

Q2 Using the voltage equation (5.1) and the fundamental torque equation (5.2), and $T = K_t i$, determine the current at steady-state speed with $V = 9$ V and $T_L = 0.003$ N $\cdot$ m. Call this current i_{d2}. See the lecture slides.

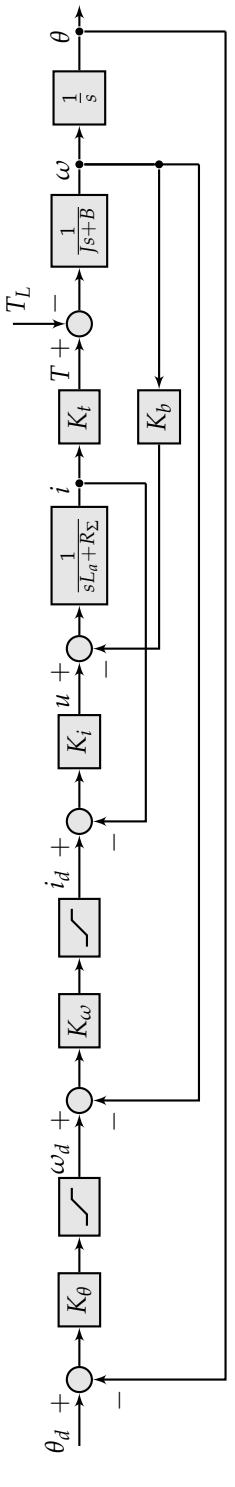


Figure 6.1: Position control of a pmdc motor. This three-loop structure is used with other types of motors too. This structure is used for tracking a given θ_d while restricting the speed or current. When only control of speed is needed while constraining the current, the outer-most loop can be removed. Abbreviation: $K_\theta \equiv K_\theta(s)$, $K_\omega \equiv K_\omega(s)$, $K_i \equiv K_i(s)$.

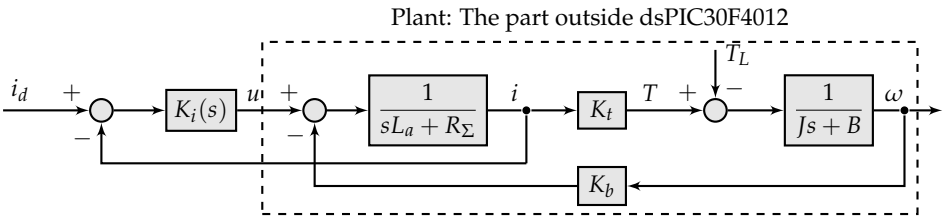


Figure 6.2: Control of current in a pmdc motor. For the design of the controller $K_i(s)$ of current, use the values of R_Σ, B that you determined in Question Q5 of Section 5.4. Note that $K_i(s)$ does not stand for an integral controller. Instead, it is the controller of the armature current i.

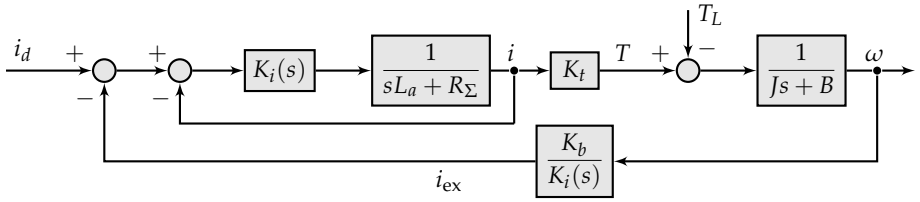


Figure 6.3: The block diagram of Figure 6.2 redrawn. We can see that i tracks $i_d - i_{ex}$, and not i_d alone. i is considered well-regulated if it tracks i_d nicely. This nice tracking will not happen while i_{ex} dominates. Therefore, we choose $K_i(s)$ to supress i_{ex}. Two choices for $K_i(s)$ are P and PI.

This value of $T_L = 0.003$ N $\cdot$ m has been arrived at as follows. The radius of the pulley where the string winds is approximately $r = 1.25$ cm. The mass of the load is in the range $m = 1.5 - 2$ kg, but we assume that it is 1.5 kg. Acceleration due to gravity $g = 9.8$ m/s^2. The gear ratio is $R_g = 62$. We have $T_L = mgr/R_g$.

This i_{d2} is the maximum value that we wish to specify as reference for the current control at $T_L = 0.003$ N $\cdot$ m. Any greater value of reference current at $T_L = 0.003$ N $\cdot$ m will require the H-bridge to apply a voltage $V > 9$ V, and thereby go into voltage saturation. We avoid saturation to have an approximately linear plant. The minimum value of i_{d2} is T_L/K_t. When we apply the load and require the motor to track a value of i_d that is less than this minimum value of i_{d2}, the load will drive the motor, rather than the motor driving the load.

Q3 Determine the TF from u to i for the PMDC motor in preparation for the design of a PI controller.

Use the values of R_Σ and B from the experiment where they were calculated from the experimentally-deterimined K_m and τ_m (See Question Q5 of Section 5.4).

As explained in Section 2.3, the armature inductance L_a is negligible.

Q4 Design a P or a PI controller for a settling time of 0.5 s.

Without using a semilog graph paper, we can still use our loop-shaping

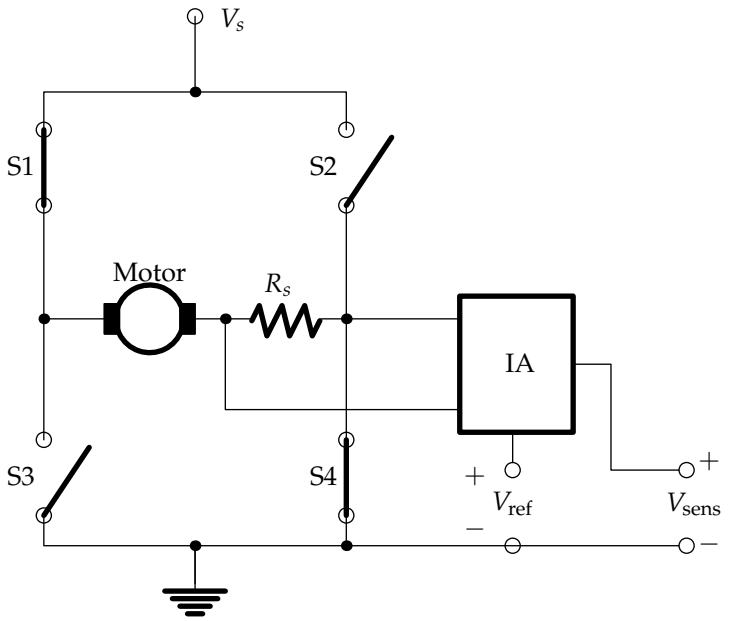


Figure 6.4: The correct placement of sensing resistor R_s to measure armature current. IA is an instrumentation amplifier. This placement of R_s correctly gives the sign of the armature current, unlike the placement of R_s in Figure 5.1. In that figure, irrespective of whether the armature current is flowing from right to left or from left to right in the armature, the current through R_s is from top to bottom. That is, that way of measuring the current shows the armature current as positive always, irrespective of the actual direction of the current.

skills to build the controller through a little bit of reasoning combined with simulation as follows.

Choice of K_P At the first instant after the control system is deployed, the error $e(0) = i_d - i(0)$ is the largest (assuming a stable control system), and, consequently, $K_P e$ is the largest at this instant. Therefore, K_P may be chosen, for example, as $K_P \times e(0) < 9$ V. For example, if $i_{d2} = 0.15$ A, then $K_P < 9/0.15 = 60$ V/A. Any value of $K_P > 60$, will drive the applied armature voltage into saturation at the initial instant of control.

Choice of K_I Based on a quick sketch of the $(K_P s + K_I)/s$, we can see that the larger the K_I, the greater the speed of response of the control system. So, K_I may be chosen as large enough for the settling time to be about 0.5 s. A simulation using the m-file provided `e5q4.m` helps.

Answer the questions in `e5q4.m` for yourself. We may quiz you on them.

Q5 Provided to you is `e5q5easysim.m`, which is a modified version of `easysim.m`. Verfiy for yourself that this file has been written correctly, and simulate the CL system of Figure 6.2 for

1. $i_d = i_{d1}$, and
2. $i_d = i_{d2}$.

Plot the i versus t and u versus t curves from parts 1 and 2.

Do the results of Q5 match those from Q4? Explain the differences.

6.5 To do in lab

Q7 Write the discretized version of the home-designed controller into `main-prog.c`. In `main-prog.c` implement the part of the block diagram of Figure 6.2 that is outside the plant. Either use $\widehat{i} = i_{\text{sens}}$, or $\widehat{i} = \frac{1}{1.8} i_{\text{sens}}$ (such as in Question Q6.3 of Section 5.4.2). The results we showed in the lecture slides are using $\widehat{i} = \frac{1}{1.8} i_{\text{sens}} - \frac{1}{1000}$, though our later tests seemed to be giving better results with the PI controller using $\widehat{i} = i_{\text{sens}}$.

Q8 Under the feedback control of Figure 6.2, take the readings of u and i for

Q8.1 $i_d = i_{d1}$ without any load on the motor. Immediately after the motor shaft seems to be rotating at constant speed, hold the shaft tightly for about a second and release.

Q8.2 $i_d = i_{d2}$ with the load provided by us tied to the pulley. Immediately after the motor shaft seems to be rotating at constant speed, hold the shaft tightly for about a second and release.

Q9 Take plots in both cases. Each plot will show i before and after application of disturbance. With what error does i track the i_d in the two cases?

Q10 Are you satisfied with the disturbance rejection? Explain.

Q11 Write down the part of the C code that you wrote for this experiment.

You will use this code in the experiment on disturbance observer.

6.6 What to check if things do not work

If you seem to have done everything correctly, and yet your control system is not working the way it should, here are some points you can check.

1. Is the electrolytic capacitor connected with correct polarity, as shown in Figure 5.3?

Thanks 2. *We thank Prof. Shyama Prasad Das for helping us formulate questions Q3 and Q4.*

Chapter 7

Experiment 6: Disturbance observer

7.1 Goal

To implement a disturbance observer (DOB) for a PMDC motor.

7.2 Background

7.2.1 Application of DOB

A conventional forward path controller such as P, PI, PID, lead, lag, etc, which is a *single degree of freedom controller* (see Figure 7.1 (a)), can reject disturbances as well as provide good performance if designed and tuned properly. The design and tuning becomes much easier with a so-called *two-degree of freedom* (2DOF) controller (Figure 7.1 (b)). There are two parts to this controller, each of which can be designed and tuned independent of the other.

7.2.2 Model of pmdc motor with well-regulated current

For the case where the developed torque T is proportional to a current i as $T = K_t i$, and where this current is well-regulated at the desired value i_d, as done in Figure 6.2, the motor-gear unit can be represented as in Figure 7.2.

For example, NASA's Mars rovers used brushed dc motors (same as PMDC motors) as their driving and steering motors [15]. For these motors, i is the armature current. Brushless dc (BLDC) motors are also an alternative, for example, in the Lunar Rover that ISRO has built in-house and that IITK has also built. For BLDC motors i is the average dc link current [16, page 769].

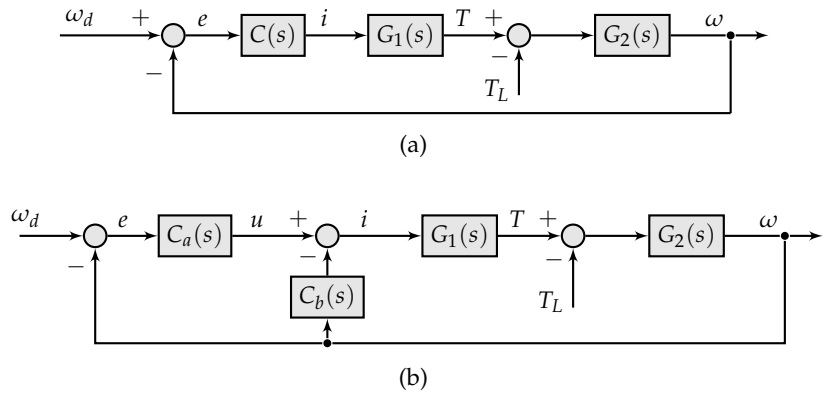


Figure 7.1: A single degree-of-freedom controller, (a), and a 2DOF controller, (b). $C_a(s)$ provides one degree of freedom while $C_b(s)$ provides the other. In the control system of (b) $C_b(s)$ helps reject T_L, while $C_a(s)$ helps track ω_d.

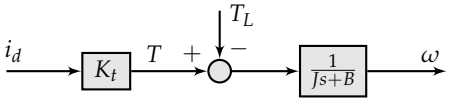


Figure 7.2: Representation of a motor unit with i well regulated at i_d.

7.2.3 DOB for a pmdc motor with well-regulated current

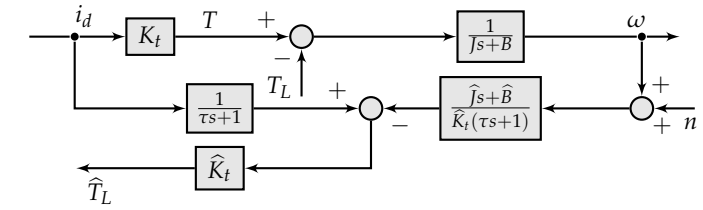
T_L can be estimated through either the OL scheme or the CL scheme of Figure 7.3. By working out the transfer functions from T_L to $\widehat{T}_L$ in these two schemes, we can see that, in the CL scheme, $\widehat{T}_L$ remains close to T_L even with variations in B, J, and K_t if τ is chosen to be sufficiently small.

Figure 7.4 shows the full block diagram of the PMDC motor with the DOB.

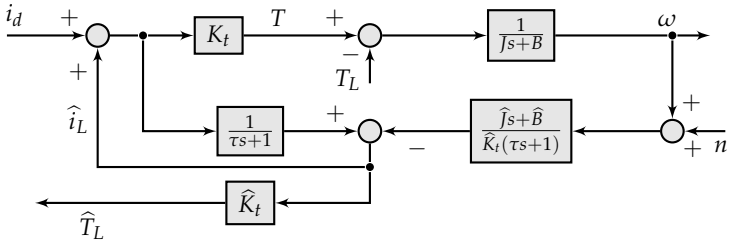
7.3 Questions

7.3.1 To do at home

1. Comparison of the DOBs of Figure 7.3.
   1.1. Form the TF from T_L to $\widehat{T}_L$ in each of the block diagrams of Figure 7.3.
   1.2. Evaluate the two TFs when τ is small. What is the value of τ?
   1.3. In which scheme is $\widehat{T}_L$ closer to T_L when of $\widehat{J}, \widehat{B}, \widehat{K}_t$ are poor estimates?
2. Run the SIMULINK file named `dob.mdl` and describe briefly the effect of each of the following changes.
   2.1. $K_\omega(s)$ being the controller you designed in Experiment 1 as opposed to the lag controller shown in `dob.mdl`.



(a)



(b)

Figure 7.3: Open-loop DOB to estimate T_L, (a), and closed-loop DOB to estimate T_L, (b), [17]. n is sensor noise. $\widehat{J}, \widehat{B}, \widehat{K}_t$ are the estimates of J, B, K_t respectively. In our experiment, we use the values of $\widehat{J}, \widehat{K}_t$ taken from Table 1.1, and the value of $\widehat{B}$ that is calculated from the unit step response of the OL motor as in Question Q5 of Section 5.4.

2.2. Injecting $\widehat{i}_L$ with a $-$ instead of $+$.

2.3. Breaking the injection of $\widehat{i}_L$.

2.4. Varying τ.

2.5. Varying the plant parameters (J, B, K_t) with the respective estimates $(\widehat{J}, \widehat{B}, \widehat{K}_t)$ kept constant at their initial values.

3. Become acquainted with the files `main-prog-exp6.c` and `easyplot.m`.

4. Verify if the controllers that appear in `dob.mdl` have been discretized correctly in `main-prog-exp6.c`.

7.3.2 To do in lab

5. Take an OL step response of your motor, and calculate R_Σ and B. You can use the latter for $\widehat{B}$. You can use $\widehat{J} = 1.34 \times 10^{-6}$ kg m^2.

6. If necessary, modify the values of B and J, and the relation between $\widehat{i}$ and i_f in `main-prog-exp6.c`.

7. Run your setup and take readings on both sides of the instant when the load steps up in the following cases.

7.1. With $\widehat{i}_L$ fed back. Plot ω and $\widehat{T}_L$ versus t

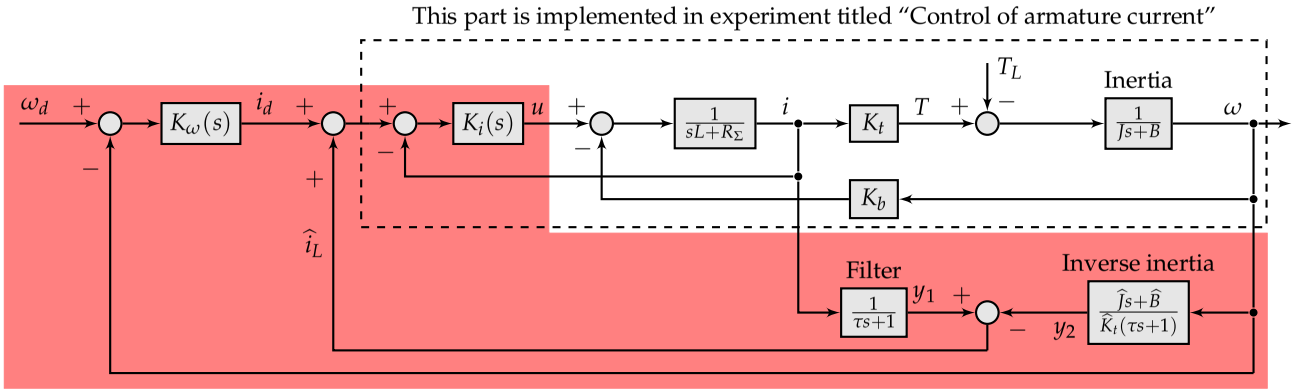


Figure 7.4: The full block diagram of the two-degree of freedom control of a pmdc motor. The first degree of freedom is provided by the DOB in the inner loop. The second degree of freedom is provided by the speed controller $K_\omega(s)$ that is in the outer loop. The shaded portion of the block diagram is implmented in dsPIC30F4012. The purpose of the filter $1/(\tau s+1)$ is to not just make the inversion of $\frac{1}{Js+B}$ practically possible, but also to improve the disturbance rejection performance of the system. See [17] for details.

7.2. With $\widehat{i}_L$ fed back. Plot ω and u versus t.

7.3. Without $\widehat{i}_L$ fed back. Plot ω and $\widehat{T}_L$ versus t

7.4. Without $\widehat{i}_L$ fed back. Plot ω and u versus t.

The m-file `easyplot.m` organizes these plots as demonstrated in the lecture. You only need to know how to use it. Make a few trial runs to obtain best possible plots.

8. How did you expect your DOB to work? How did it actually work?

7.4 Programs provided

You are provided with the following files:

1. `dob.mdl`: This SIMULINK file simulates the full block diagram of Figure 7.4.
2. `main-prog-exp6.c`: This file implements the block diagram in C code. This file implements the block diagram of Figure 7.5.
3. `easyplot.m`: This m-file helps plot the results read into the PC into `terminal.log` by `terminal.exe` from the μC.

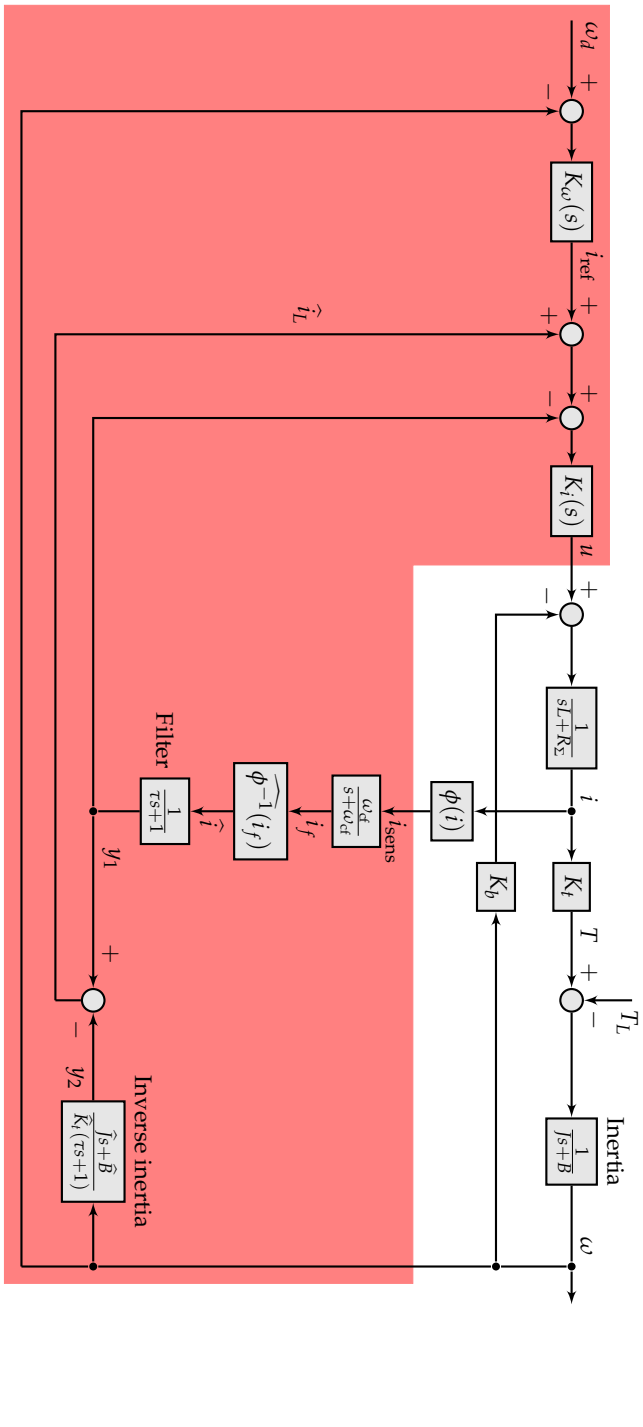


Figure 7.5: The full block diagram of the two degree of freedom control system as implemented in our lab. The shaded portion of the block diagram is implemented in `main-prog-exp6.c`.

Chapter 8

Experiment 7: Disturbance observer without feedback of current

8.1 Goal

To build and test a pmdc motor speed control system that uses a disturbance observer (DOB) but does not use the feedback of armature current.

8.2 Background

In Experiment 6, we design a DOB using the feedback of armature current i. In the present experiment, we build a slightly modified version of that DOB that does not use the feedback of i.

Figure 8.1 (b) shows the block diagram of Figure 7.4 simplified. Figure 7.4 is reproduced for convenience as Figure 8.1 (a). The advantage of the simplified block diagram is that it does not need the measurement of current if we have the measurement of speed. The disadvantage is that we do not have $\widehat{T}_L$. In applications where we need $\widehat{T}_L$, we need to implement the scheme of Figure 8.1 (a), and not the simplified one.

From the SIMULINK model, that we we see that if we replace $\frac{1}{\tau s+1}$ with 1 in the path from i to y_1, then there is no visible change in the behavior of the overall system. Thus, after performing this replacement, the overall block diagram can be redrawn as in Figure 7.4 (b).

In our lab, we will implement the scheme of Figure 7.4 (b). One reason for this is that scheme (a) needs us to have measurement of current i. This quantity, in our experiments, is of the order of a few tens of milliamps and contains significant amount of noise. In the dual-motor ball-beam setup that we built

in the Networked Control Systems Lab, we used a capacitor across the sensing resistor to remove this noise, and an instrumentation amplifier to boost the resultant signal. In the Control Systems Lab, however, we have not yet implemented this filter-amplifier combination. So, measurements of current are not yet accurate enough for feedback.

8.3 Questions

8.3.1 To do at home

1. Play with the SIMULINK file named `dobgm.mdl` to acquaint yourselves with the two-degree of freedom control system with the DOB. Some items of interest are
   1.1. What is the effect of $K_\omega(s)$ being a PI controller as opposed to a lag controller?
   1.2. What is the effect of $K_I(s)$ being a PI controller as opposed to a proportional controller?
   1.3. What is the effect of injecting $\hat{i}_L$ with a $-$ instead of $+$?
   1.4. What is the effect of break the injection of $\hat{i}_L$?
   1.5. What is the effect of varying τ?
   1.6. What is the effect of varying the plant parameters (J, B, R, K_t) with the respective estimates $(\hat{J}, \hat{B}, \hat{K}_t)$ kept constant at their initial values?
2. Write the equations that describe the overall closed-loop system of Figure 7.4 (b).
3. Discretize these equations.
4. Check if your discretized equations match what we gave you in the attached m-file `expt6.m`.
5. If everything matches, then run the `expt6.m` in GNU Octave.
6. Write the C code of the controllers that you will implement in the lab.

8.3.2 To do in lab

7. Take an OL step response of your motor, and calculate R_Σ and B. You can use the latter for $\hat{B}$. You can use $\hat{J} = 1.34 \times 10^{-6}$ kg m^2.
8. Modify the values in the C code from Question 6, if necessary.
9. Write your C code into the appropriate place in the `main-prog.c`
10. Run your setup and take readings on both sides of the instant when the load steps up. Provide the following two plots as subplots one below the other: u versus t, and ω versus t.

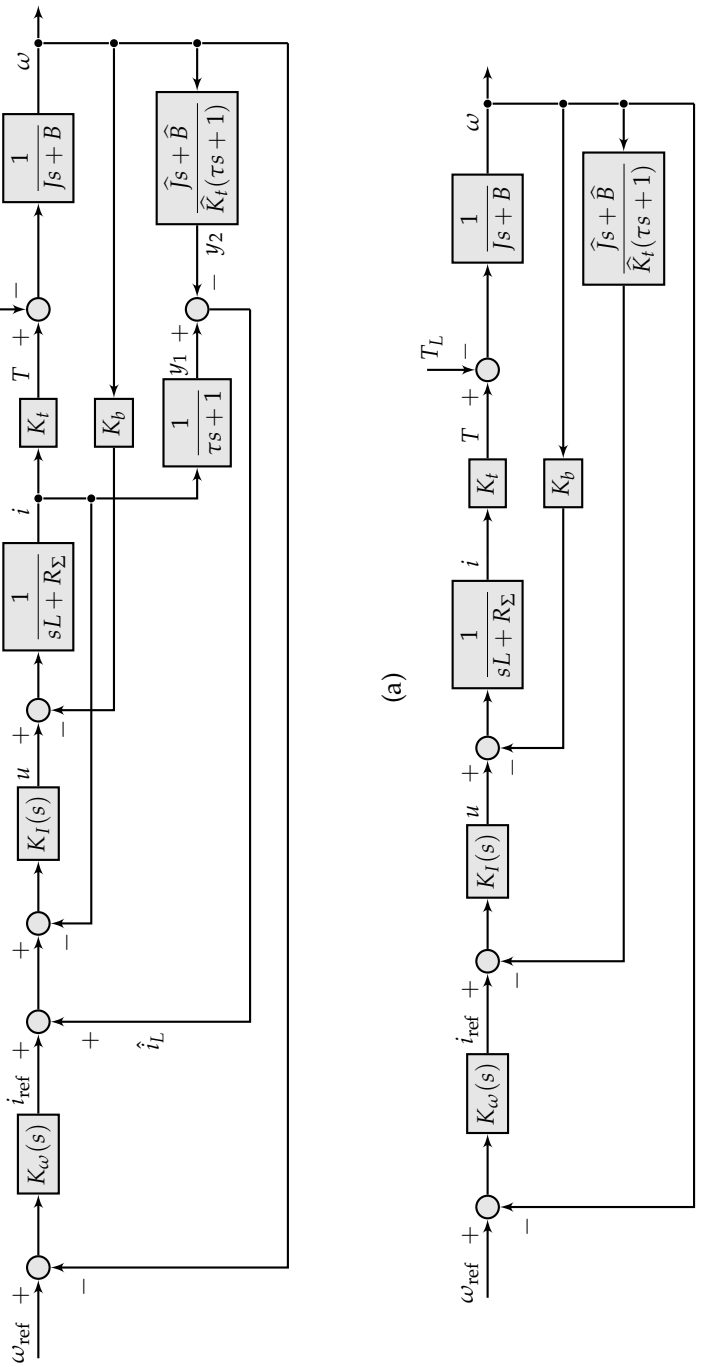


Figure 8.1: (a) The overall block diagram of the two-degree of freedom control of a PMDC motor, and (b) The block diagram of (a) simplified to remove the feedback of current. This simplification is done by setting the gain from i to y_1 equal to 1 instead of $1/(\tau s+1)$. The advantage of scheme (b) is that it does not need the measurement of current if we have the measurement of speed. The disadvantage is it does not give us $\widehat{T}_L$; therefore, this scheme does not really realize a DOB. In applications where we need $\widehat{T}_L$, we need to implement scheme (a).

11. How did you expect your DOB to work? How did it actually work?

8.4 M-files

```
%% expt6.m
clc; clear all; close all;
format long

T = 0.002; tend = 2; t = 0:T:tend;
La = 0.00077; % in H
Jm = 13.0e-7; % in kg m^2
Jg = 0.4e-7;  % in kg m^2
Kt = 0.0255;  % in Nm/A
Kb = 0.0255;  % in V/(rad/sec)
J = Jm+Jg;

%% practical values from step response
B = 5.41e-6;
Ra = 31;

%% Simulation
Ci = 100; % inner controller (current controller)

% let speed controller Cw = (a1*s+a2)/(s+a3)
b0 = 0.001157; b1 = 0.0222; a1 = 0.02218;
Wr(1:tend/T+1) = 100;
wf = 100; %% filter frequency in rad/sec

% TL = 0.00316;  % for 2 Kg 0.01*9.8*2/62
tTL = 1; % time when the TL is applied
TL(1: tTL/T) = 0;
TL(tTL/T+1:tend/T+1) = 0.00316;

% Initialization
W = 0; xw = 0; xid = 0; xir = 0;
for k = 1:tend/T+1
 %  Speed controller
 ew(k) = Wr(k) -W(k); % speed error
 Iref(k) = (b1-b0*a1)*xir(k) + b0*ew(k);
 xir(k+1) = (1-a1*T)*xir(k) + T*ew(k);

 % estimation of Ihat = Im-IL
 Ihat(k) = (B*wf/Kt - J*wf^2/Kt)*xw(k) + J*wf/Kt*W(k);
 xw(k+1) = (1-T*wf)*xw(k) + T*W(k);

 % Inner controller (Current controller)
 u(k) = Ci*(Iref(k) - Ihat(k));
```

```
 % motor speed calculation
 W(k+1) = ( 1 - B*T/J - Kt^2*T/Ra/J )*W(k) + Kt*T/J/Ra*u(k) -T/J*TL(k);
end

%% plot the response
subplot(3,1,1), plot(t, TL); grid;
title('T_L [in Nm]');
subplot(3,1,2), plot(t, u); grid;
title('Controller output voltage [in V]');
subplot(3,1,3), plot(t,Wr, t,W(1:tend/T+1)), grid;
title('Speed \omega [in rad/sec]'); legend('\omega_r', '\omega')
```

Chapter 9

Experiment 8: PMDC motor modeling, identification, position control

9.1 Goals

Modeling and identification (using OL step response) of PM DC motor. Design using root locus and implementation of position controller for both. Comparison of results.

9.2 Introduction

In this experiment we will study the theoretical and practical aspects of the control of the angular displacement of the shaft of a PM DC motor using a microcontroller. The block diagram of the setup is as shown in Figure 1.1. A dsPIC30F4012 micro-controller is used as the platform on which to implement our digital controller, and H-bridge four-quadrant DC chopper is used to drive the DC servo motor. Please see the first two chapters of this manual for the details of the hardware and the software platform. Follows a list of questions for this experiment.

9.3 Mathematical model of DC servo motor

Since the inductance L_a is very small, we may neglect L_a. Then the TF between the motor shaft position $\theta(s)$ and the armature voltage $V(s)$ is

$$\frac{\theta(s)}{V(s)} = \frac{K_m}{s(\tau_m s + 1)} \tag{9.1}$$

where, K_m is the motor gain constant and τ_m is the motor time constant. These are as in Equation (2.1).

The parameters of the motor, gear, and encoder are provided in Table 1.1.

9.4 Dead zone in the V_m versus u characteristic

The dead zone in the V_m versus u charaacteristic of Figure 1.7 did not affect much in the control of the speed of the DC motor as the speed was needed to be at a large value and u quickly becomes large.

In the case of control of position of the shaft of the shaft of the DC motor, however, u may become small when the desired position is reached. If u becomes smaller than 2 volts when $V_s = 12$ V, then V_m will be zero, and the position control system will become unresponsive. To overcome this situation we have two alternatives:

1. Use an integral component in the controller.
2. Add the following code in `main-prog.c` before or after the if condition used for limiting the duty ratio.

```
if(u<0&&u>-2)
       u =  u - 2;
else if(u>0&&u<2)
       u =  u + 2;
```

9.5 Questions

9.5.1 To do at home

Q1 Determine the physics-based mathematical model of the DC servo motor whose parameters may be found in the attached data sheets.

Q2 Design using root locus techniques a controller of at least first order and at most second order to control the angular displacement of the shaft of the given motor for the following time domain specifications: $e_{ss} \leq 2\%$, $t_s = 0.5$ s, $M_p \leq 20\%$.

For root locus-based design, please see the class notes from EE250_Fall2010.

Q3 Simulate the continuous-time controller designed above using Matlab.

If the closed-loop system performance in simulation is not as desired, you may need to redesign your controller.

Q4 Discretize the continuous-time controller.

Q5 With the discretized version, perform a simulation of digital control of the continuous-time plant using the m-file provided in Experiment 1.

Do you think that your digitally-controlled closed-loop system will be stable in practice? Will it provide in practice the same performance as did the continous-time version in simulation?

Q6 Write the digital controller in C.

9.5.2 To do in lab

Q7 Write a code to give a step input to the given motor. Run the setup in OL mode.

Q8 Identify the system parameters K_m and τ_m by using the OL step response.

Q9 For the identified model, redesign your controller using root locus in MATLAB.

Q10 With the discretized version of the above-redesigned controller, perform a simulation of digital control of the continuous-time plant using the m-file provided in Experiment 1.

Q10 Program the home-designed digital controller and run the setup. Record and plot the necessary data.

Q11 Program the lab-designed digital controller and run the setup.

Q12 Compare the following results:

1. those using the controller designed for the physics-based model and obtained on Matlab,
2. those using the controller designed for the physics-based model and obtained from the experimental setup,
3. those using the controller designed for the identified model and obtained on Matlab,
4. those using the controller designed for identified model and obtained from the experimental setup.

Q13 Conclusions:

1. Is the physics-based model a good match to the plant? If not, what do you think we have ignored that has lead to the difference?
2. What are the skills you learned from this experiment?
3. Would you have preferred to learn a different skill set from the control lab? If yes, which skills?
4. How could we have organized this experiment differently to make it more meaningful to you?

Chapter 10

Experiment 9: Encoderless speed control of PMDC motor using compensation of plant nonlinearity

This experiment is modification of Experiment 4, and is based on [2].

10.1 Questions

10.1.1 To do at home

Q1 Using the values of K_m and τ_m determined from the OL step response in Experiment 1, determine the values of B and R_Σ from the equations

$$K_m = \frac{K_T}{R_\Sigma B + K_T K_b}, \tau_m = \frac{R_\Sigma J}{R_\Sigma B + K_T K_b}.$$

Use the values of J, K_T, K_b from Table 1.1. Do not use the value of B calculated in Section 1.6.

TIP: Write a small (4 – 6 lines) GNU Octave code for doing this calculation as you may be required to do the calculation once again in the lab. Bring the Octave code to the lab

Q2 Write down the controller that you designed in Experiment 1.

Q3 Provided for this pre-lab assignment is a C-file `main-prog-exp9.c`. Examine this code and answer the following questions:

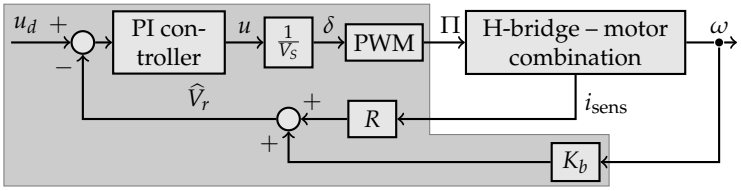


Figure 10.1: A control system that helps automatically measure the input-ouput characteristic of the H-bridge.

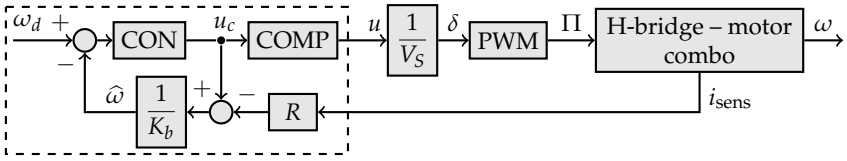


Figure 10.2: The encoderless speed control system with compensation of H-bridge nonlinearity.

Q3.1 Write down the transfer function, with numerical values, of the PI controller used in Figure 10.1. (Note: $R = R_\Sigma$ in the figures in this assignment)

Q3.2 When you modify this C-file to use on your setup, which values in the block diagram will you modify in the C-file?

Q3.3 Values of which variables in the block diagram are sent to the PC by this C-file?

Q4 Provided for this pre-lab assignment is an m-file `readplot_exp9.m` and a .log file name `exp9saurav.log`. Run the m-file on the .log file and answer the following questions.

Q4.1 Write down the equation that you would use in the Figure 10.2 as the compensator. (NOTE: You will obtain your own compensator in the lab)

Q4.2 Write the C-code implementation of the part of the block diagram that is in the dashed box.

Q4.3 Where will this C-code go — into `main-prog-exp9.c` or `main-prog.c`?

Q4.4 For how many seconds was `main-prog-exp9.c` run to generate the data in `exp9saurav.log`?

10.1.2 To do in lab

Verify that in your experimental setup the lead from the sensing resistor R_s is connected to CN4 input pin 1 as shown in Figure 5.3, in addition to the usual connections for speed control.

Uncomment the following parts of `main-prog.c`

```
  //  IV = AD_value();  // Read voltage across Rs=4.7ohm.
  //  IV = 5*(511 + IV)/1022; // Convert signed to unsigned.
```

and

```
  //  Is = IV/4.7;     // Convert voltage to current.
  //  IF = (1-5.0*T)*IF + 5.0*T*Is; // Low-pass filter.
```

There are other parts of `main-prog.c` that you will uncomment at your discretion.

Q5 Write a code to apply a step input to the motor. Run the setup in OL mode. Identify the system parameters K_m and τ_m by using the OL step response. **[$\sim$ 10 min]**

Q6 If K_m and τ_m are different from those you saw in Experiment 1, calculate the values of R_Σ and B as you did in Q1. Use the Matlab code of Q1 for this calculation. **[$\sim$ 5 min]**

Q7 Use the controller from Experiment 1. Control the motor using feedback of speed. Sketch ω versus t and u vs. t in the table provided. Use $\omega_{\text{ref}} = 100$ rad/s in Q7, Q9, Q12. **[$\sim$ 10 min]**

Q8 Use the controller from Experiment 1. Control the motor using feedback of current with $\widehat{\omega} = \frac{u - R_\Sigma i}{K_b}$. Sketch ω vs. t, $\widehat{\omega}$ vs. t, and u vs. t in the table provided. **[$\sim$ 15 min]**

Q9 Answer the following questions: **[$\sim$ 10 min]**

Q9.1 Why are the steady state values of ω in Q7 and in Q8 not the same?

Q9.2 Why are the steady state values of ω and $\widehat{\omega}$ not equal in Q8?

Q10 Burn the C-file `main-prog-exp9.c` into the μC. Take care to modify the value of R. This file implements the block diagram of Figure 10.1 and returns the data for u_d and u. Record this data into a log file using `terminal.exe`. **[$\sim$ 10 min]**

Q11 Use the m-file `readplot_exp9.m` to process the data given out by `main-prog-exp9.c`. **[$\sim$ 10 min]**

Q11.1 Write down the polynomial generated by the m-file.

Q11.2 Sketch the compensator generated by this m-file in the below space.

Q12 Insert the compensator polynomial at the appropriate place in `main-prog.c` and repeat Q8. That is, Figure 10.2 needs to be constructed. Sketch ω vs. t, $\widehat{\omega}$ vs. t, and u_c vs. t in the table provided. **[$\sim$ 20 min]**

Q13 Do you think that the polynomial given out by `readplot_exp9.m` is a good compensator for the nonlinearities introduced by the H-bridge? **[$\sim$ 1 min]**

Q	Var	Plot
Q7	ω	
Q7	u	

Q	Var	Plot
Q8	ω	
Q8	$\widehat{\omega}$	
Q8	u	

Q	Var	Plot
Q12	ω	
Q12	$\widehat{\omega}$	
Q12	u	

Software used

The Controls Lab has the following software on Microsoft Windows XP.

MPLAB IDE and MPLAB30 C compiler

The specific programs that are used in our lab are

`MPLAB_IDE_8_86.zip`
`mplabc30_v3_30c_windows.exe`

`MPLAB_IDE_8_86.zip` can be downloaded from

`http://ww1.microchip.com/downloads/en/DeviceDoc/`
`MPLAB_IDE_8_86.zip`

MPLAB C30 compiler can be found by searching on `www.microchip.com` for "mplab c30", without the quotes. Exists "MPLAB C Compiler for Academic Use" that is used in this lab. It can be downloaded from here:

`http://www.microchip.com/stellent/`
`idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en536656`

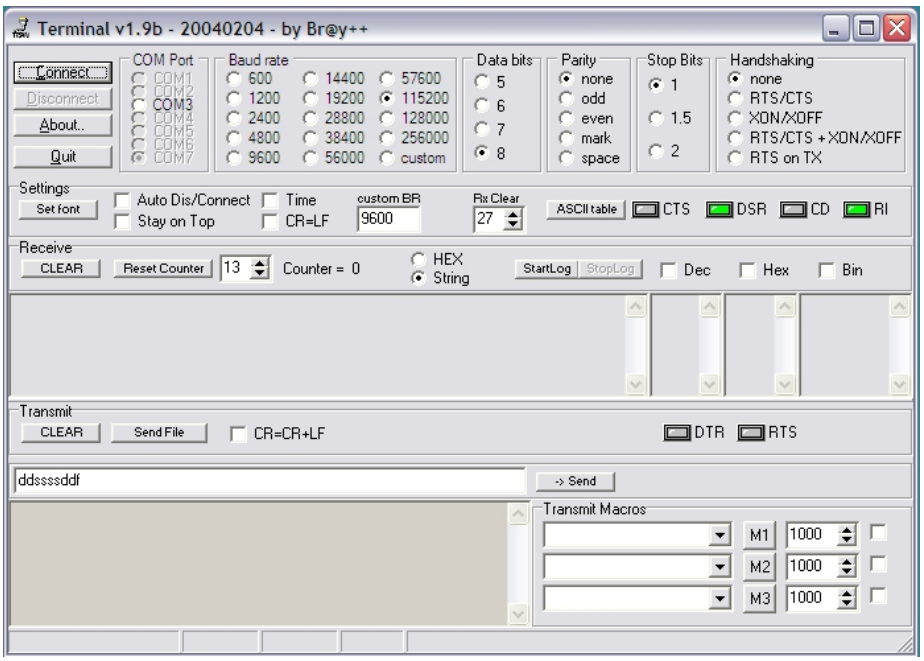
Microchip also has `MPLAB_IDE_8_87.zip` on their site. This 8.87 version may work too. It can be downloaded from

`http://www.microchip.com/stellent/`
`idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=`
`en019469&part=SW007002`

Occassionally, when burning to the dsPIC from MPLAB IDE fails, we use PICKIT 2 software, which came packaged as `PICkit 2 v2.61.00 Setup A.zip` on PICkit 2 programmer's CD.

Terminal.exe

The version of Bray's `terminal.exe` used in the lab is version 1.9b 20040204.



GNU Octave

We have been using versions of GNU Octave >= 3.2.4. GNU Octave can be downloaded from the website named Octave-Forge that has the URL `http://octave.sourceforge.net/`. On this page, there is the following link to a Windows installer of GNU Octave

`https://sourceforge.net/projects/octave/files/Octave%20Windows%20binaries/`

with installation instructions that are easy to implement.

Bibliography

[1] M. Gunasekaran and R. Potluri. Low-cost undergraduate control systems experiments using microcontroller-based control of a dc motor. *IEEE Transactions on Education*, 55(4):508 – 516, Nov. 2012.

[2] Kumar Saurav and Ramprasad Potluri. Sensorless speed control of a permanent magnet dc motor by compensating the plant nonlinearities. In *IEEE International Symposium on Industrial Electronics (ISIE)*, pages 1–4. IEEE, 2013.

[3] Frank L.Lewis and Vassilis L.Syrmos. *Optimal Control*, pages 67–68. Wiley Interscience, 2 edition, 1995.

[4] Gene F. Franklin, J. David Powell, and Michael L. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley, 3 edition, 1998.

[5] Katsuhiko Ogata. *Modern Control Engineering*. Pearson Education, fourth edition, 2002. Available in India.

[6] Andrzej Banaszuk [People in Control]. *Control Systems, IEEE*, 32(3):27 –31, June 2012.

[7] Tore Hägglund and Karl J. Åström. Automatic tuning of PID controllers. In William S. Levine, editor, *The Control Handbook*, pages 817 – 826. CRC Press LLC, 1996.

[8] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. Pearson Education, fourth edition, 2002.

[9] Graham C. Goodwin, Stefan F. Graebe, and Mario E. Salgado. *Control System Design*. Prentice-Hall, 2001.

[10] SGS-Thomson Microelectronics. TDA7274: Low-voltage dc motor speed controller, 1988. Data sheet.

[11] W. Zhan. Sensorless speed control for dc permanent magnet motors. In *Proceedings of the ninth IASTED International Conference on Control and Applications*, pages 116 – 120. 2007.

[12] Centent Company. CN0055 & CN0055B: dc to dc negative resistance speed control, 2011. Datasheet.

[13] A. Cordeiro, D. Foito, and M. Guerriro. A sensorless speed control system for an electric vehicle without mechanical differential gear. In *IEEE MELECON 2006*, pages 1174 – 1177. Benalmadena, Malaga, Spain.

[14] Gopal K. Dubey. *Fundamentals of electrical drives*. Narosa, 2001.

[15] Maxon Motors. 5 years on mars. `http://www.maxonmotor.com/media_releases_5619.html`, 2009.

[16] H.A.A. Toliyat and G.B. Kliman, editors. *Handbook of Electric Motors*, chapter Electronic Motors. CRC Press, second edition, 2004.

[17] T. Umeno and Y. Hori. Robust speed control of dc servomotors using modern two degrees-of freedom controller design. *IEEE Transactions on Industrial Electronics*, 38(5):363 – 368, October 1991.