Abstract

This project seeks to demonstrate a technique for evaluating the velocity of an electromagnetic motor without utilizing external sensors. In order to do so, the back-emf voltage of the motor is extracted and processed, and it is proven that there is a linear relationship between the voltage and the motor’s angular velocity. In the process, this project utilizes a MSP430 microprocessor to control a pulse-width modulation driven motor.
# Contents

1. Introduction 3
2. Background/Theory 3
3. Completed Project Design 4
4. Results 6
5. Discussion/Conclusion 8
6. Appendix: Code 9
1 Introduction

The goal of this project was to find a method in which to accurately evaluate the velocity of a electromechanical motor. In particular, the project focused on capturing the velocity without relying on external sensors. Instead, the project took an indirect approach by obtaining the voltage produced by the Back-EMF of the motor, and proving that it is directly proportional to the angular velocity. As a result, accurate velocity measurements can be made without additional sensors and potential additional sensory errors, allowing for quicker response and control feedback.

2 Background/Theory

This project uses a electromagnetic motor. Hence, from Faraday’s law, we can safely state that the back emf voltage of the motor, $\varepsilon$, can be described by:

$$\varepsilon = \frac{d\Phi_B}{dt}$$

Where $\Phi_B$ is the magnetic flux of the system.

The magnetic flux is defined by the surface integral:

$$\Phi_B = \int \int \vec{B}d\vec{A}$$

where $\vec{B}$ is the magnetic field strength.

In the motor, the magnet turns relative to the conducting coils around it, allowing us to state in general form that the emf voltage is proportional to the angular velocity.

$$\varepsilon = \int \vec{B}d\vec{A} = K \int d\vec{x} = K\omega$$

for some constant $K \in \mathbb{R}$.

In this project, the motor is controlled using a circuit configuration called a H-Bridge. The circuit utilizes transistors to direct the flow of current through the motor. In order to evaluate the effective measurement of motor speed, the motor only needs to be driven in one direction.
In the depiction above, the circuitry can be set such that the upper left transistor and lower right transistor are turned permanently off, allowing only current to pass through from right to left.

3 Completed Project Design

In order to evaluate the velocity of the motor without impeding the overall operation of the motor, the motor was set to coast for a small portion of the time between PWM drive, allowing the back-EMF voltage to exit the transient stage and provide an accurate voltage reading without causing noticeable physical change in the motor operations. Consider the following image of the H-Bridge.
As mentioned before current is only allowed to flow from right to left by shutting off the left PMOS transistor and the right NMOS transistor. This is done using the code shown below:

```c
P2DIR |= (P1 | N1 | P2 | N2); // set pins controlling transistors to output
P2SEL |= (N1); // set NMOS to be controlled by
// pin 2.2 is TAl.1
P2SEL2 = 0x00;
P2OUT &= ~N2; // N2 is always low (set off)
P2OUT |= P2; // P2 always high (set off)
```

In the code below, the PWM is set to a default duty cycle, which it runs for a long period of time. Then the right PMOS transistor is turned off, and the left NMOS transistor is turned fully on, allowing the motor to coast. In this short period of time, the computer converts the back-emf voltage and converts it to a digital value, before reverting back to PWM drive.

```c
while (1) {
    // Do this continuously
    P2OUT &= ~P1; // TAlCCR1=pwm1val;
    unsigned int i;
    for (i=0; i<50000; i++);
    P2OUT |= P1; // TAlCCR1=16;
    for (i=0; i<100; i++); // wait for transient state
    ADC10CTL0 |= ADC10SC; // Start conversion
    vval = ADC10MEM; // Get A/D conversion result;
    for (i=0; i<1000; i++);
}
```

The following graphical depiction of the voltage reading process. The green line indicates an oscillating PWM driven motor, which is stopped for a short period of time (milliseconds) and generates solely the back-emf voltage motor with essentially no decay.
Note: The back-EMF motor voltage at maximum speed capacity had a range between 8 - 12 volts. The MSP430 can only operate in the 0 - 3.3 V inputs range. In order to generate a digital value for the voltage through the MSP430, the motor voltage was sent through a voltage divider with a resistor ratio of 3:1, thereby reducing MSP430 input voltage to the appropriate range, as depicted in the graph above.

4 Results

We now proceed to validate that the obtained voltage data can be linearly fitted to the actual angular velocity of the motor, thereby giving us an interface on which to direct the motor in future cases. First the motor velocity controlled through PWM was verified to be appropriately linear.
We note that below a duty cycle of 20 %, the motor has not exceeded a base threshold voltage, thereby unable to move the physical motor. Therefore values below 20 % were not considered. Next, the digital value output processed from the AD motor was examined and checked for linearity as well.

Here, the AD value appears strictly linear, until the duty cycle exceeds 75 %, where it flattens out around 500. This is discussed in the subsequent section. In the meantime, the linear portion of AD conversion values were evaluated relative to the actual angular velocity to produce a conversion factor.
Using the coefficients, we have a confirmed linear relationship between motor velocity and the back-emf voltage.

5 Discussion/Conclusion

The first and foremost issue to be addressed is the AD conversion value stalling above 75% duty cycles. It is not likely to be a numerical limit issue, as the maximum obtainable voltage input is 2.81 V, well below the MSP430 enforced limit of 3.3 V. Additionally, the remaining data strongly indicated a linear relationship, and confirms that the signal processing is not affected by external noise. Further tests on the motor’s properties and performance may yield a rationalization.

Possible extensions include, but are not limited to:

1. The current project is used only to test and verify the potential to evaluate the velocity of a electromagnetic motor without the use of external sensors. The motor velocity is currently controlled through pulse-width modulation. If the system were to be used to enable motor control, it would require a more robust interface. In such an interface, a user could simply have abstract functionality to read and set the motor velocity and/or position freely.

2. Writing code for reversible motor drive, as the motor currently only operates in one direction. This would simply require adding a secondary functionality in the code to drive the current through the motor in the reverse direction, requiring a mirror set up of the default H-Bridge set up.

3. Use of a more robust motor, allowing greater range of operation and accuracy.
6 Appendix: Code

See next page.
#include <msp430.h>

/*
 * main.c
*/

/* Define bits for inputs
 * Only use these with pin 2 */
#define P2 BIT1 // define transistors
#define N1 BIT2
#define P1 BIT3
#define N2 BIT4

void coast();
void run();

volatile int vval, pwm1val;

void main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
    P1OUT |= 0x01; // Light LED as a check

    /* set processor clk to 1 MHz */
    DCOCTL = 0x00;
    BCSCTL1 = CALBC1_1MHZ;
    DCOCTL = CALDCO_1MHZ;

    // PWM related set up
    BCSCTL1 |= XT2OFF;
    BCSCTL3 = XT2S_0 + LFXTI1S_2 + XCAP_1;
    BCSCTL2 &= ~0x06;
    BCSCTL2 = DIVS_1; // Set divider for SMCLK = 2
    P1DIR = 0x11; // P1.4 output;
    P1SEL = 0x10; // P1.4 is SMCLK
    TA1CTL = TASSEL_2 + MC_1+ID_1; // Timer A sourced from SMCLK
    TA1CCTL1 = OUTMOD_7; // CCR1 reset/set (PWM)

    /* set up A/D converter:
     * -ADC10ON- turn ADC on
     * -ENC- conversion enabled
     */
    ADC10CTL1 = INCH_5; // Set to channel 5
    ADC10CTL0 = ADC10ON | ENC; //Turn ADC on, enable conversions
    TA1CCR0 = 16-1; // CCR0 = PWM Period-1 (15625Hz w/ SMCLK=250kHz)
    pwm1val = 8; // PWM val is 0 to 16, 8 is 50%

    _BIS_SR(GIE); // Enable general inputs
    P2DIR |= (P1 | N1 | P2 | N2); // set pins controlling transistors to output
    P2SEL |= (N1); // set NMOS to be controlled by
                   // pin 2.2 is TA1.1
    P2SEL2 = 0x00;
main.c

58   P2OUT &= ~N2; // N2 is always low (set off)
59   P2OUT |= P2; // P2 always high (set off)
60   P2OUT |= N1; // N1 is always high (set on)
61
   while (1) { // Do this continuously
58   P2OUT &= ~P1;  TA1CCR1=pwm1val;
64   unsigned int i;
66   for (i=0; i<50000; i++);
67   P2OUT |= P1;  TA1CCR1=16;
68   for (i= 0; i < 50; i++); // wait for transient state
69   ADC10CTL0 |= ADC10SC; // Start conversion
70   vval = ADC10MEM; // Get A/D conversion result;
71   for (i=0; i<1000; i++);
72
74
75   } //********End, PWM executed*************************************************************************/
76
77 }
78
79 /*********************************************************************************************/
80
81
This code utilizes the raw data from the motor and checks for linearity.

% Raw Data:

% PWM set values
pwmval = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16];

% AD Conversion values
ADvalue = [208, 248, 280, 318, 347, 380, 412, 448, 474, 500, 505, 505, 505, 505];

% Radians/per second (Motor Velocity)
RPS = [0.527, 1.043, 1.764, 2.196, 2.963, 3.529, 4.106, 4.456, 4.908, 5.235, 5.660, 6.100, 6.613, 6.981];

% Conversions:

% Conversion to percent active
pwmpercent = (pwmval./16) * 100;

% Secondary set discounting higher AD Conversion cap
ADvalue2 = [208, 248, 280, 318, 347, 380, 412, 448, 474, 500];

% Corresponding set for Motor Velocity
RPS2 = [0.527, 1.043, 1.764, 2.196, 2.963, 3.529, 4.106, 4.456, 4.908, 5.235];

hold on
plot(pwmpercent, ADvalue);
title('AD Conversion Value vs. Percent Active')
xlabel('% PWM')
ylabel('ADvalue')
figure(1);
plot(pwmpercent, RPS);
title('Radians/Sec vs. Percent Active')
xlabel('% PWM')
ylabel('Rad/s')
figure(2);
plot(ADvalue2, RPS2)
AD Conversion Value vs. Percent Active

AD Conversion vs. Radians/Sec
xlabel('ADValue');
ylabel('Rad/s');
hold off
Radians/Sec vs. Percent Active

AD Conversion vs. Radians/Sec