Real-Time Adaptive PID Controller Using the TMS320C31 DSK

Jianxin Tang
Div. of Electrical Engineering
Alfred University
Alfred, NY 14802

Rulph Chassaing, Walter J. Gomes III
Dept. of Electrical and Computer Engineering
University of Massachusetts Dartmouth
North Dartmouth, MA 02747

Abstract

This paper addresses real-time adaptive PID (Proportional-Integral-Derivative) controller using the TMS320C31 DSK. With an adaptive PID controller, the C31 automatically updates the PID controller parameters to achieve the desired system output. The controller is applied to a DC motor system for speed control. Different adaptation rates were tested. Test results show that PID parameters converged to expected values and motor speeds reach desired output speeds corresponding to different setpoints. Future work includes extending the results to the C6x.

I. Introduction

Due to the recent and remarkable progress in power electronics and microelectronics it is possible to apply modern control technology to the area of motor and motion control. The use of digital signal processors (DSPs) has permitted the increasingly stringent performance requirements and fast, efficient, and accurate control of servo motor and motion control systems. DSPs, such as the TMS320C31 (C31) from Texas Instruments, are currently used for a wide range of applications from controls and communications to speech processing. They continue to be more and more successful because of available low-cost support tools. DSP-based systems can be readily reprogrammed for a different application.

The term adaptive system has a variety of specific meanings, but it usually implies that the system is capable of accommodating unpredictable environmental changes, whether these changes arise within the system or external to it [1]. This concept has a great deal of appeal to the systems designer since a highly adaptive system, besides accommodating environmental changes, would also accommodate engineering design errors or uncertainties and would compensate for the failure of minor system components, thereby increasing system reliability.

The C31-based $99 DSK includes Texas Instruments' C31 floating-point digital signal processor, and an analog interface circuit (AIC) chip with A/D and D/A

converters, input (anti-aliasing) and output (reconstruction) filters, all on a single chip. The A/D converter can accommodate two multiplexed inputs, a key feature that is required for adaptive control with an error signal input and a reference signal input. The DSK also includes an assembler, a debugger, and many application examples [2].

This paper addresses real-time DC motor speed control with an adaptive PID controller using the C31 DSK. A PID controller has one fixed pole at the origin of the complex *s*-plane (or at 1 in *z*-plane) and two flexible zeros. Since the pole is fixed, all we have to do is to make the two zeros adaptive. Code Explorer was used to run the program on the C31 DSK and monitor initial and final values of the PID parameters. Test results show that the adaptive process converges, the final PID parameters are reasonable, and the system response is as expected.

In Section II, design of the adaptive scheme for the PID controller is discussed. Implementation of the adaptive PID controller using the C31 is addressed in Section III. Test results of the motor system and comparison with theoretical calculations are presented in Section IV. Finally conclusion and future work are given in Section V.

II. Design of the Adaptive Scheme for the PID Controller

The block diagram of the adaptive control system is as follows:

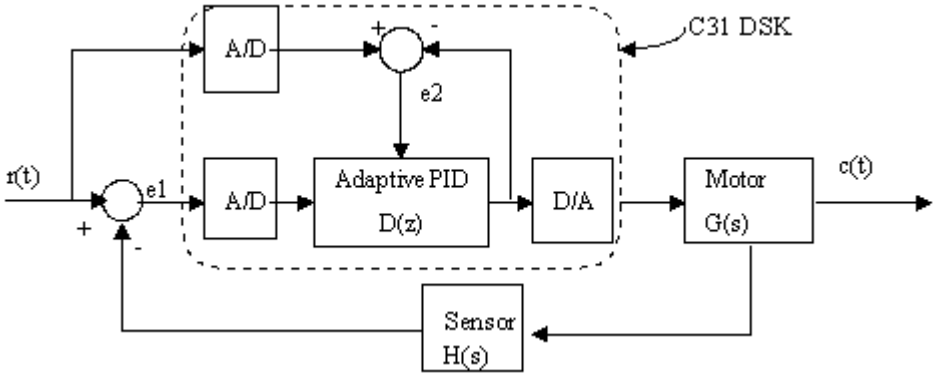

Fig. 1. The adaptive motor control system

In Fig. 1, r(t) is the input (or set point), c(t) is the output, D(z) is the digital PID controller, G(s) is the plant transfer function, and H(s) is the sensor transfer function. The adaptive control scheme consists of two parts: the first part is using initial or updated PID parameters, the controller will be taking in input samples, processing them, and sending them out to the motor through the D/A converter; the second part is updating the controller parameters. This process continues until the error signal e2 is approaching zero.

A digital non-adaptive PID controller has the following form [3]:

$$D(z) = K_P + K_I \frac{T}{2}\frac{z+1}{z-1} + K_D \frac{z-1}{Tz}$$

$$= \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 - z^{-1}} \quad (1)$$

with

$$a_0 = K_P + \frac{K_I T}{2} + \frac{K_D}{T}$$

$$a_1 = -K_P + \frac{K_I T}{2} - \frac{2K_D}{T}$$

$$a_2 = \frac{K_D}{T},$$

where K_P, K_I, and K_D are the proportional, integral, and derivative parameters of the controller, and T is the sampling period. It is clear from equation (1) that the PID controller has one fixed pole and two flexible zeros. For programming convenience, let

$$D(z) = D(z)\frac{M(z)}{M(z)} = \frac{Y(z)}{X(z)}$$

where X(z) and Y(z) are the input and output of the controller in the z-domain, respectively, then

$$Y(z) = (a_0 + a_1 z^{-1} + a_2 z^{-2})M(z) \quad (2)$$

and

$$X(z) = (1 - z^{-1})M(z). \quad (3)$$

Using the inverse z-transformation,

$$y(k) = a_0 m(k) + a_1 m(k-1) + a_2 m(k-2) \quad (4)$$

and

$$m(k) = x(k) + m(k-1). \quad (5)$$

As mentioned in section I, the C31 DSK has two multiplexed inputs. One is the primary input and the other is the auxiliary input. Denoting the primary input as IOPRI and the auxiliary as IOAUX, the detailed portion of the C31 DSK in Fig.1 is shown in Fig. 2, where E1 and y are input and output of the PID controller, respectively.

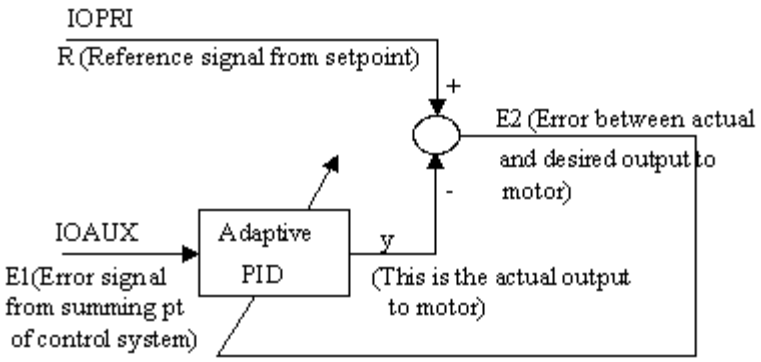

Fig. 2. Detailed portion of the adaptive PID controller

E2 is then:

$$E2(z) = R(z) - Y(z) = R(z) - \left[\frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 - z^{-1}}\right] E_1(z) \tag{6}$$

A quadratic objective function is created based on minimizing E2 with respect to the controller parameters:

$$J(a_0, a_1, a_2) = \tfrac{1}{2}(E2)^2$$

$$= \tfrac{1}{2}\left[R(z) - \left(\frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 - z^{-1}}\right) E1\right]^2 \tag{7}$$

Taking the first order partial derivative with respect to the controller parameter a's:

$$\frac{\partial J}{\partial a_0} = -2R\left(\frac{1}{1 - z^{-1}}\right) E1 + 2\left(\frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 - z^{-1}}\right)\left(\frac{1}{1 - z^{-1}}\right) E1^2 \tag{8a}$$

$$\frac{\partial J}{\partial a_1} = -2R\left(\frac{z^{-1}}{1 - z^{-1}}\right) E1 + 2\left(\frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 - z^{-1}}\right)\left(\frac{z^{-1}}{1 - z^{-1}}\right) E1^2 \tag{8b}$$

$$\frac{\partial J}{\partial a_2} = -2R\left(\frac{z^{-2}}{1 - z^{-1}}\right) E1 + 2\left(\frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 - z^{-1}}\right)\left(\frac{z^{-2}}{1 - z^{-1}}\right) E1^2 \tag{8c}$$

Using the first order approximation, the negative gradient is basically the product of reference signal R and error signal E1. Tests show that using E1 and E2 actually has a better convergence Therefore, a modified gradient method was used as the search direction in updating the PID parameters. This also agrees with the general adaptive mechanism mentioned in [4]: new parameter = old parameter + (bounded step size) X (function of input) X (function of error). More specifically

$$a_n(k+1) = a_n + \beta e_2(k) e_1(k-n) \quad n = 0,1,2 \tag{9}$$

III. Implementation of the adaptive PID Controller Using the C31 DSK

Following is the assembly language program ADAPTPID.ASM implementing the adaptive PID controller.

```
*ADAPTPID.ASM - Adaptive PID controller using the C31 DSK
        .start   ".text", 0x809900 ;starting address of text
        .start   ".data", 0x809C00 ;starting address of data
        .include "AICCOM31.ASM"    ;include AIC comm routines
        .entry  BEGIN              ;start of code
        .text                      ;assemble into text
BEGIN   LDP     @COEFF_ADDR        ;init to data page 128
        CALL    AICSET             ;initialize AIC
        LDI     @ERF_ADDR,AR6      ;error function address
        LDI     @XNB_ADDR,AR2      ;AR2=bottom addr of input xn to filter
        LDI     LENGTH,BK          ;BK=length of circular buffer
PID     CALL    IOAUX              ;get input error signal
        FLOAT   R6,R3              ;stage input
        STF     R3,*AR2++%         ;store newest sample
        LDI     @COEFF_ADDR,AR0    ;AR0 points to coefficients address
        LDI     @DLY_ADDR,AR1      ;AR1 points to addr of delay samples
        MPYF3   *AR0++,*AR1++,R0   ;b[0]*dly[0]=b1u(n-1)
        MPYF3   *AR0++,*AR1--,R1   ;b[1]*dly[1]=b2u(n-2)
||      SUBF3   R0,R3,R3           ;input-b[0]*dly[0];R3=x(n)-b1u(n-1)
        MPYF3   *AR0++,*AR1++,R0   ;a[1]*dly[0];R0=a1u(n-1)
||      SUBF3   R1,R3,R2;u(n)=xn-b[0]*dly[0]-b[1]*dly[1]=x(n)-b1u(n-1)-b2u(n-2)
        MPYF3   *AR0++,*AR1--,R1   ;a[2]*dly[1]=R1=a2u(n-2)
        ADDF3   R0,R1,R3          ;a[2]*dly[1]+a[1]*dly[0];R3=a1u(n-1)+a2u(n-2)
        LDF     *AR1,R4            ;dly[0];R4=u(n-1)
||      STF     R2,*AR1++          ;dly[0] = dly; u(n-1)updated to ->u(n)
        MPYF3   R2,*AR0--,R2       ;dly*a[0];R2=a0u(n);point to a2 to adapt
||      STF     R4,*AR1++          ;dly[1] = dly[0];u(n-2)->u(n-1) to update
        ADDF3   R2,R3,R3           ;controller out;y=a0u(n)+a1u(n-1)+a2u(n-2)
        FIX     R3,R7              ;convert to integer for output
        CALL    IOPRI              ;get reference desired signal d
        FLOAT   R6,R4              ;R4=reference desired signal d
        SUBF3   R3,R4,R0           ;R0=error signal=d-y
        MPYF    @BETA,R0           ;ERR function=e*beta
        STF     R0,*AR6            ;store error function
       CALL    ADAPT               ;call ADAPT subroutine
        BR      PID                ;branch back/repeat with new input sample
;ADAPTATION ROUTINE
ADAPT   MPYF3   *AR6,*AR2++%,R0    ;error function*x(n-(N-1)) ->R0=erfx(n-2)
        LDF     *AR0,R3            ;w(N-1) -> R3=a2
        ADDF3   R3,R0,R2           ;w(n-1-i)+erf*x(n-(N-1-i));R2=a2+erfx(n-2)
        STF     R2,*AR0--          ;store/upgrade a2 coeff
        MPYF3   *AR6,*AR2++%,R0    ;erf*x(n-(N-1-i))->R0=erfx(n-1)
```

```
        LDF     *AR0,R3             ;load subsequent w(k) ->R3=a1
        ADDF3   R3,R0,R2            ;w(n+1)=w(n)+erf*x(n);R2=a1+erfx(n-1)
        STF     R2,*AR0++(2)        ;store/upgrade a1 coeff;then points to a0
        MPYF3   *AR6,*AR2++%,R0     ;R0=erfx(n);erf*newest sample in circ buffer
        LDF     *AR0,R3             ;R3=a0
        ADDF3   R3,R0,R2            ;R2=a0+erfx(n)
        STF     R2,*AR0             ;store/upgrade a0
        RETS
        .data  ;b[0]       b[1]        a[1]        a[2]       a[0]
COEFF  .float  -1.0000E+0, 0.0000E+0, -0.0000E+0, 0.0000E+0, 1.0000E+0
DLY        .float  0, 0               ;init delay var for each stage
COEFF_ADDR .word   COEFF              ;address of COEFF
DLY_ADDR   .word   DLY                ;address of DELAY
XNB_ADDR   .word   XN+LENGTH-1   ;bottom addr of cir buffer for error signal xn
ERF_ADDR   .word   ERR_FUNC           ;address of error function
ERR_FUNC   .float  0                  ;initialize error function
BETA       .float  10E-14             ;rate of adaptation constant
AICSEC     .word   162Ch,1h,244Ah,73h ;AIC config data, Fs = 16/2 kHz
LENGTH     .set    3                  ;Length of circular buffer for xn
           .brstart "XN_BUFF",16      ;align on 16-word boundary
XN         .sect   "XN_BUFF"          ;section for buffer
           .loop   LENGTH             ;loop length (3) times
           .float  0                  ;initialize buffer to zero
           .endloop                   ;end of loop
           .end                       ;end
```

Fig. 3. ADAPTPID.ASM for the adaptive PID controller

This program is based on the combination of the IIR filter program and the adaptive filter for noise cancellation program in [2]. It consists of two major subroutines. Subroutine PID takes in input samples and calculates outputs of the PID controller, using existing PID parameters. It also calculates the error between desired output and actual output of the system. Subroutine ADAPT updates the PID parameters a_0, a_1, a_2, using equation (9). Two inputs are required in this application, available on the AIC on board the DSK. While the primary input IOPRI is through an RCA jack, a secondary input IOAUX to the AIC is available on the DSK board from pin 3 of the 32-pin connector JP3. The secondary input is enabled from the setting in AICSEC in the program. The AIC communication subroutine in AICCOM31.ASM, included in the ADAPTPID.ASM program, are set so that the extended precision registers R6 and R7 are used for input and output, respectively. The program was assembled using the assembler included in the C31 DSK package, and was run using Code Explorer. For initial conditions, b_0 and b_1 are fixed at -1 and 0, respectively, because they represent the fixed pole of the PID controller (see equation (1) in section II). Therefore they are not updated. Parameters a_0, a_1, a_2 are selected to be 1, 0, 0. Note that one can not select zeros for all three a's otherwise the control loop would be open.

Figure 4 shows the C-version of the assembly-coded PID adaptation program.

```
/*ADAPTPID.C -Real-Time Adaptive PID controller algorithm */
#include "aiccomc2.c"                          /*Include AIC comm routines*/
int AICSEC[4] = {0x162C,0x1,0x244A,0x73 }; /*AIC config data, Fs = 16/2 kHz */
#define beta 10e-14                            /* Rate of convergence */
float a[3]={1,0,0};                            /* Numerator coefficients */
float b[2]= {-1,0} ;                           /* Denominator coefficients*/
float dly[2] = {0}, input[2]={0},yn=0;         /* Global variables */
void pid(float xn)      /* Standard PID controller based on a real-time IIR algorithm*/
{
 float un;
 un = xn - b[0] * dly[0] - b[1] * dly[1];      /* Calculate yn*/
 yn = a[2]*dly[1] + a[1]*dly[0] + a[0]*un;
 dly[1] = dly[0];                              /* Update the delay samples*/
 dly[0] = un;
}
void adapt(float error_function, float xn)     /* Adaptation routine */
{
 a[2] = a[2] + (error_function * input[1]);    /* Update the numerator coefficients */
 a[1] = a[1] + (error_function * input[0]);
 a[0] = a[0] + (error_function * xn);
 input[1]=input[0];                            /* Update the adaptors delays */
 input[0]=xn;
}
void c_int05()          /* Interrupt routine to service incoming and outgoing CODEC data */
{
 float control_error,desired,error_func;
 control_error = ((float) UPDATE_AUXSAMPLE(yn)); /* Get sample from IOAUX*/
 desired = (float) UPDATE_PRISAMPLE(yn);  /* Get sample from IOPRI */
 pid(control_error);                           /* Call standard real-time PID */
 error_func = (desired - yn) * beta;           /* Calculate error function */
 adapt(error_func,control_error);              /* Call adaptation routine */
}
void main()
{
 AICSET_I();                                   /* Setup CODEC and wait for interrupt */
 while (1) {}                                  /*infinite loop*/
}
```

Fig. 4. C-version of Adaptive PID

IV. Test Results

The test was performed on a motor system of Figure 1. The actual circuit diagram is shown in Figure 5.

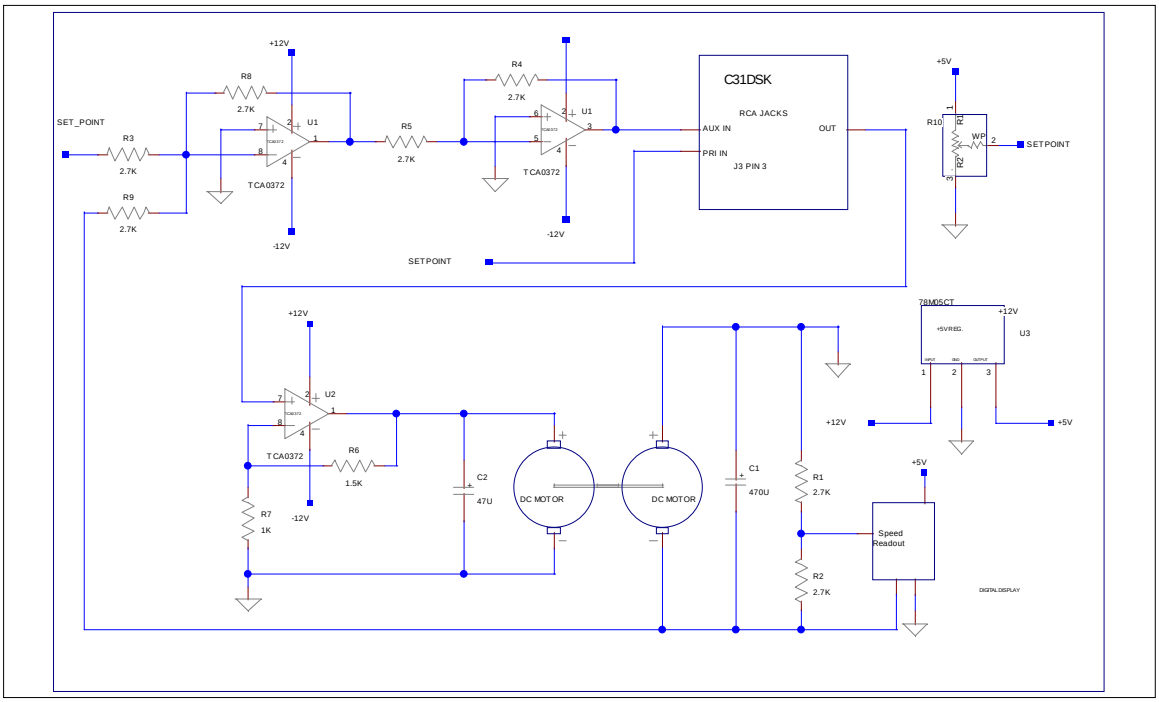

Fig. 5. Circuit diagram of the system

The mathematical model of the motor is derived first. Using a data acquisition system, the rise time of the motor is about 0.24 seconds (or a time constant of 0.08 seconds). Also, the DC gain (steady-state gain) of the motor is about 5.8. Based on this information, the motor model is calculated to be

$$G(s) = \frac{A}{s+B} = \frac{72.5}{s+12.5}$$

where 1/B is the time constant and A/B is the DC gain. In other words, the motor has a pole at –12.5. The setpoint applied to the system is 2 volts, corresponding to an output speed of 1200 rpm. The step size (adaptation rate) in equation (9) is set to be $2.5*10^{-12}$.

The final PID parameters are:

$$a_0 = 0.66443, \qquad a_1 = -0.33278, \qquad a_2 = -0.33114. \tag{10}$$

With this set of coefficients, the steady-state error (E_1 at steady state) is about 0.025 volts (should be zero ideally), or 15 rpm. This indicates that the adaptation process is doing very well in converging to the optimal parameters. Almost identical system response was obtained when the PID parameters of equation (10) were used in a non-adaptive PID program in [5].

In order to make sure that the PID parameters in equation (10) make sense. The following theoretical analysis was performed. Parameters $a_{0,}\, a_1, a_2$ are converted into k_P, k_I, k_D using the inverse bilinear transform [5]. With a sampling frequency of 8 KHz, it turns out that k_P =0.99532, k_I =4.08, and k_D = -0.0004125. Parameter k_D can be approximated with 0, which also makes sense. This is because the D-part is mainly for reducing overshoot but there is no overshoot for a first-order system. With k_D omitted, the only zero remained is $-k_I / k_P = -4.08$. With the adaptive PID controller added to the system, the root locus plot is shown in Figure 6 below. Notice that the two loci are located on the real axis, indicating no overshoot in the system. Because of the addition of a pole at the origin from the adaptive PID controller, however, the system type is increased by 1 and the steady-state error is reduced to zero.

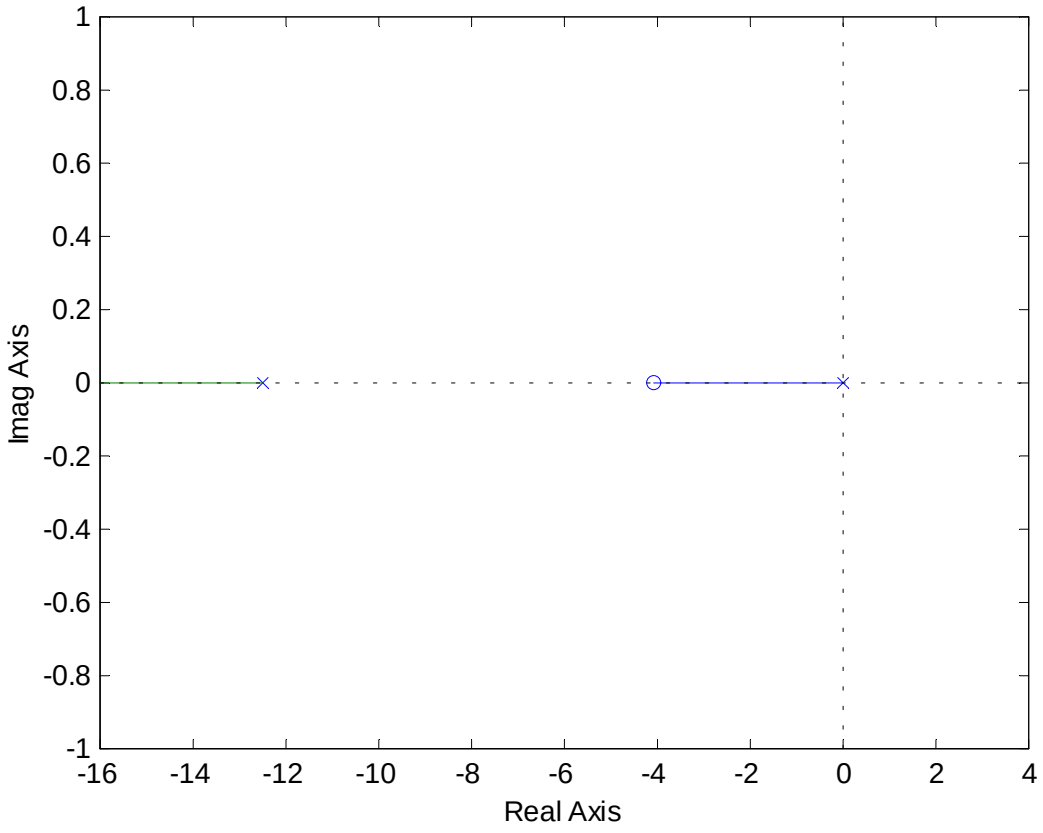

Fig. 6. Root locus plot of the system

V. Conclusion and Future Work

An adaptive PID controller has been designed and implemented using the C31 DSK. The controller is applied to a DC motor system for speed control. Test results show that controller parameters have converged to optimal ones and system output is as expected. The system is stable with different adaptation rates. Future work includes improving the control circuitry to have forward and reverse rotation, extending the system to position control, and the results to the C6x-based processor.

References

1. K. Astrom, B. Wittenmark, *Adaptive Control*, Addison Wesley, 1995.
2. R. Chassaing, *Digital Signal Processing Laboratory Experiments Using C and the TMS320C31 DSK*, Wiley, 1999.
3. C. Phillips, H. Nagle, *Digital Control Systems Analysis and Design*, Prentice Hall, 1995.
4. Y. Dote, *Servo Motor and Motion Control Using Digital Signal Processors*, Prentice Hall, 1990.
5. J. Tang, R. Chassaing, *"PID Controller Using the TMS320C31 DSK for Real-Time DC Motor Control,"* Proc. of the 1999 Texas Instruments DSPS Fest, http://www.ti.com/sc/docs/general/dsp/fest99/poster/ltangchassaing.pdf, Houston, Texas, August, 1999.
6. B. Widrow, S. Stearns, *Adaptive Signal Processing*, Prentice Hall, 1985