

Real Time Embedded Systems

Project Report

Entitled

Real Time Speed Control of a DC Motor using C505C

Microcontroller

By

Sukumar Kamalasan

ID NO. 999-90-8068

The University of Toledo

May 2001

1.0 Objective

To develop a PID controller using C505C Microcontroller for Real-Time DC motor Speed Control.

2.0 Overview

This Project aims at the development of PID (Proportional Integral Derivative) Controller using C505C Microcontroller. The rapid progress in microelectronics and Real-time Micro Controllers in recent years has made it possible to apply modern control technology in all the areas especially in control. The use of Real time Control is an area, which needs to be assessed due to the revolution and advancement of microprocessor chip technology.

This project mainly focuses on using the power and speed of C505C and modelling a Motor Control application. The peripheral devices and the internal memory along with serial communication interface are the backbone of this application.

The following peripherals have been used for this application.

- Timer
- Counter
- A/D Converter
- Serial Communication Interface (USART)
- CAN

Finally the objective of Motor Control has been assessed.

3.0 Design of the Project

The project design basically consists of two parts

1. Development of a suitable tested model of a DC motor PID controller
2. Development of an algorithm for simulating the Controller in C505C

3.1 Development of DC Motor PID Controller Model

A block diagram of the system is shown in Fig.1.

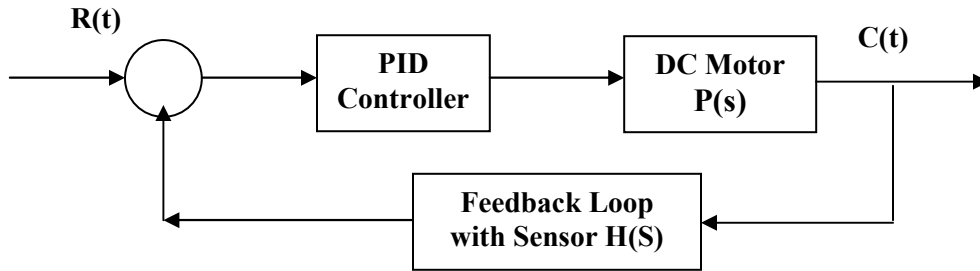


Figure 1: System Block Diagram.

Based on the above block diagram the following technical details need to be developed.

- Development of a transfer function wherein the DC Motor can be well modelled.
- Development of the PID structure with a Digital model
- Getting the various suitable parameters for the Motor and the PID equations.

The calculated values of the motor parameters and the equation for the same are developed based on the Research paper by **Jianxin Tang et.al** [1].

Normally, the output is measured and fed to the PID Controller every sample time. Depending on the process, a sample time (T_s) of 0.1 sec is taken for the simulation.

The curve generated as the Simulink output for the input parameters is shown in Fig.2. These parameters have been taken for the project. This Graph shows that the output of the Motor, which is the motor speed, has been controlled by the PID Controller and remains constant at 15 rps within 0.5 to 1 sec for the respective parameters. This will become the benchmark for our application.

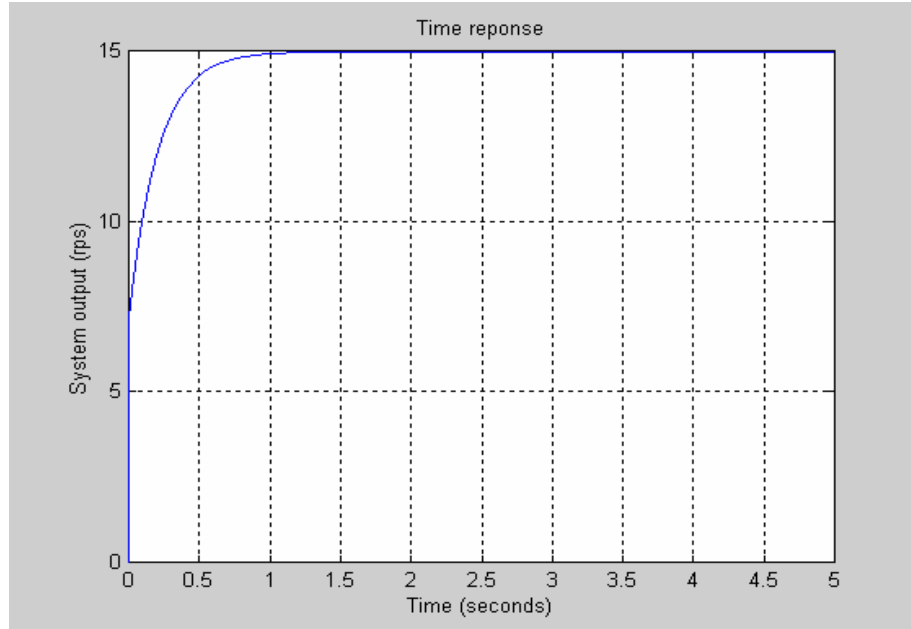


Figure 2: Time Response Characteristics for the System Output from the machine

Further, calculating the Equations from the proposed simulation we get,

DC Motor Equation in digitized form

$$G_{DC}(S) = \frac{22}{s(s+4)}$$

PID Controller

$$D(Z) = K_p + K_i \frac{T}{2} \frac{(z+1)}{(z-1)} + K_d \frac{(z-1)}{Tz}$$

Where K_p , K_i and K_d are the proportional, Integral and Derivative Controllers respectively.

Converting these equations in time varying form we get

PID_Controller = {1.001, -0.996, 0, 0, 0} / {1, -1, 0, 0, 0}; (which shows the PID Controller value for the time span as desired)

DC Motor Equation = {0, 1.8151, -1.8060, 0, 0} / {1, -1.6703, 0.6703, 0, 0}; (which shows the motor characteristic which needs to be controlled)

3.2 Development of an algorithm for simulating the Controller in C505C

The actual implementation of the PID Controller was done using the 'C' program. DAVE and Micro Vision packages were used to load the code into the C505C.

The algorithm development and the Design followed are mentioned in the flowchart shown in Fig.3.

The main steps in designing the algorithm are:

- Develop the System Model Equations and Controller
- Provide an Array for the previous 4 values of the System Output and Control.
- Declare all the data types and initialise them.
- Find out the Controller Output and the corresponding System Output.
- Get the change in Speed. (Here the change in the Potentiometer acts as change in the speed.
- Get the deviation in the Potentiometer change and the find the Error in speed.
- Calculate new controller output with the controller equation and the Error.
- Apply Control to the process and control the system output.
- Stabilize the System Output and check the Effect of the Controller

4.0 Flow Chart

The flowchart of our project is shown in figure 3.

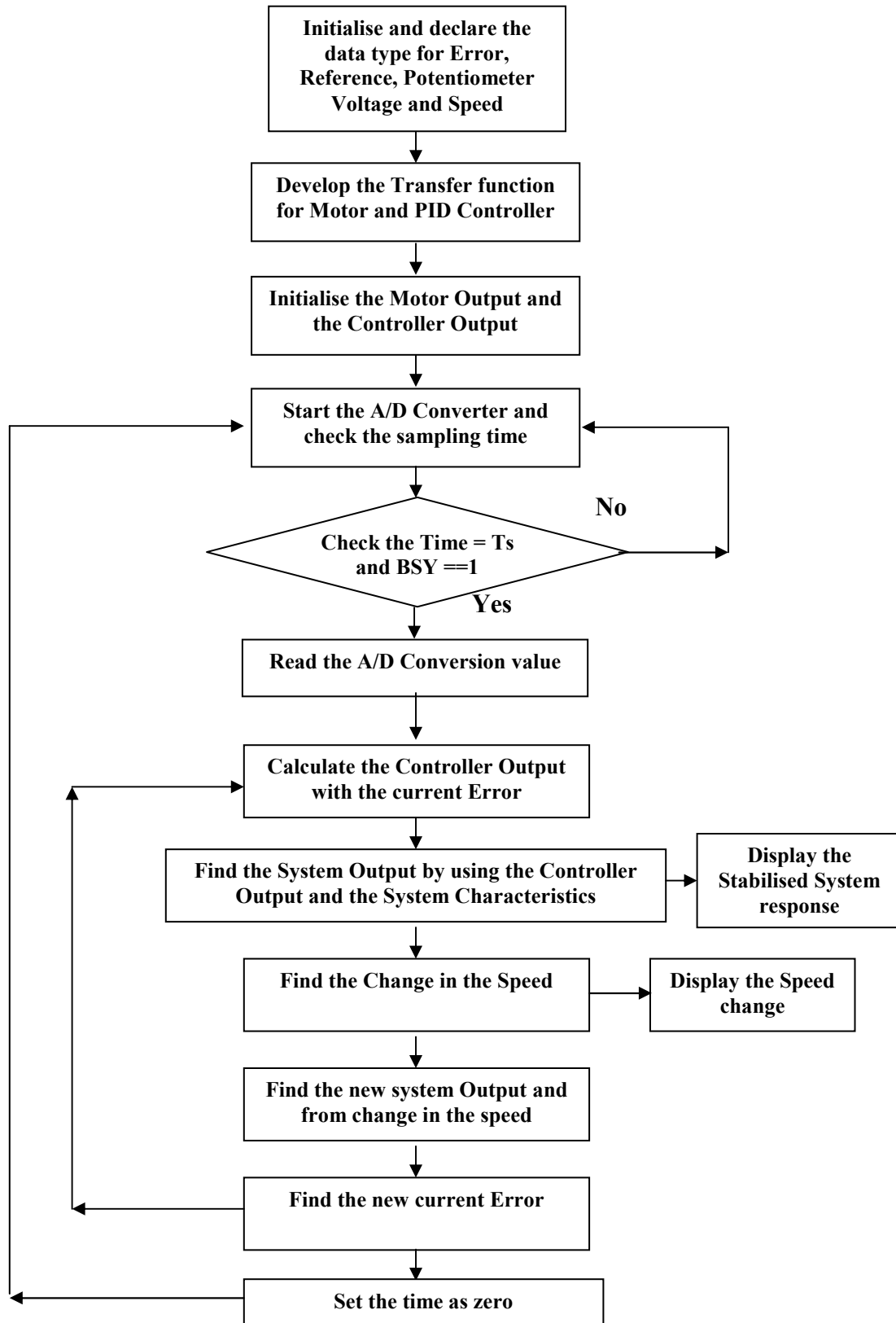


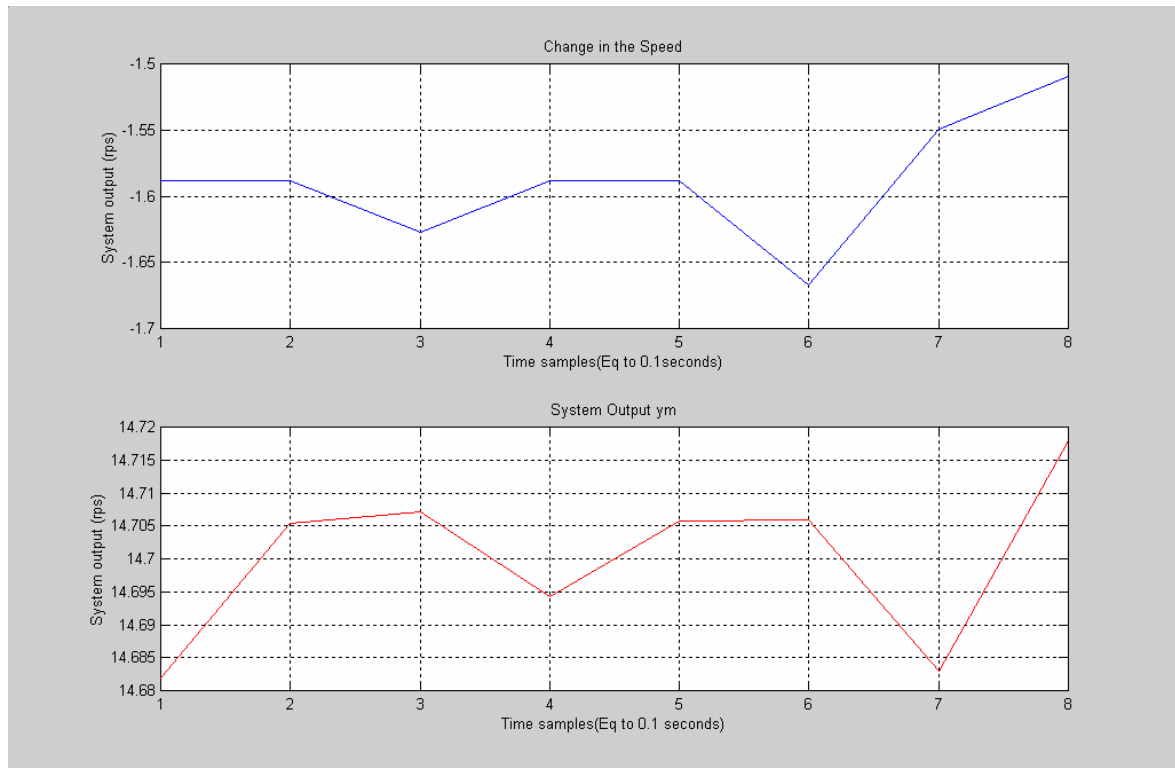
Figure 3: Project Flowchart

5.0 Project Execution

Based on the design and development of the algorithm, the project was executed as follows:

- The PID Controller and the motor model equations were initially integrated into one microprocessor and the results were displayed in another microprocessor using CAN. This was done in order to concentrate on getting the PID controller to work properly before complicating the project.
- A change in load speed was simulated within the DC motor model equations by changing the settings of a potentiometer. The change in the Potentiometer settings corresponds to a change of ± 5 rps from the rated speed (15 rps).
- The Controller and the motor model equations were then separated and loaded into two different microprocessors
- The new motor speed (system response) was transmitted via CAN to the second microprocessor which contains the PID controller.
- This new speed is compared with the reference (required) speed i.e. 15 rps. The error is fed to the PID controller algorithm.
- The controller output is transmitted back to the microprocessor that contains the motor model in order to correct the motor speed to 15 rps.
- The hyper terminal was used to print the system response, controller output, and error signal at both the motor end as well as the controller end.

6.0 Simulation Results



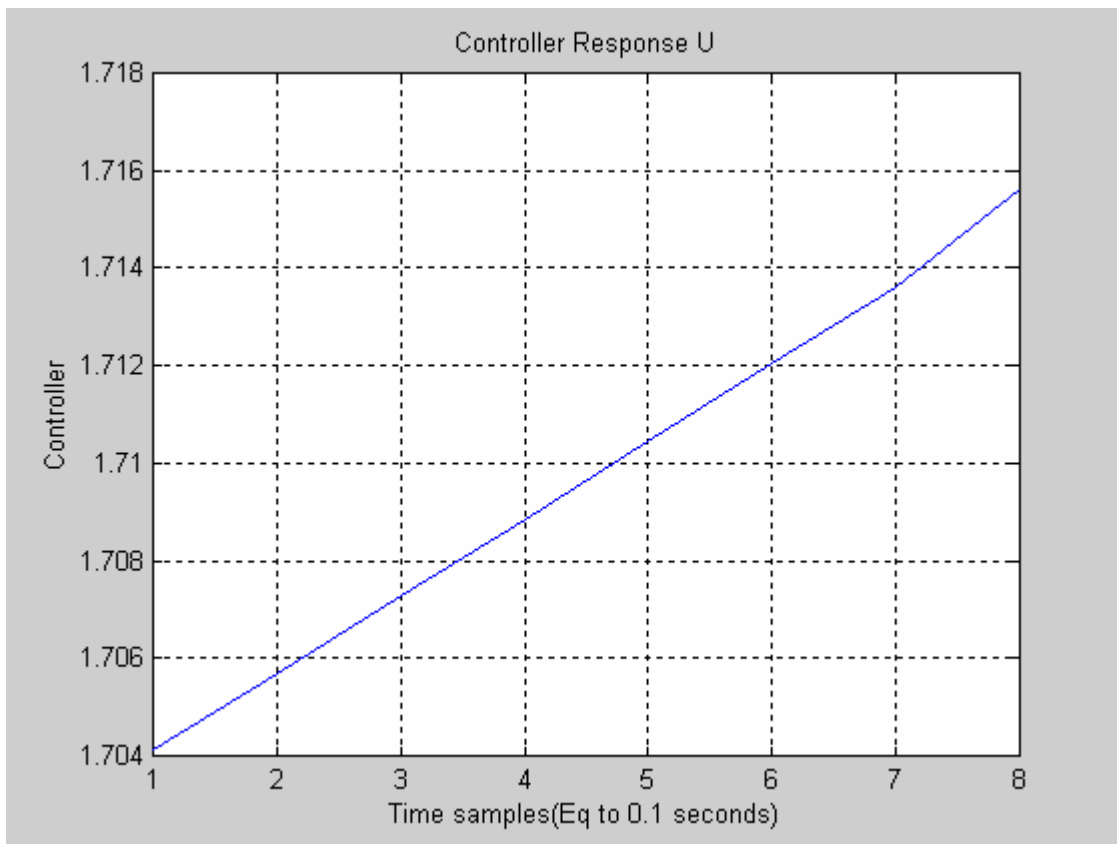


Figure 4: Controller Response

Output Response

```

Pot_Speed =-1.588235
System Response =14.681980
Controller Response =1.704113
Pot_Speed =-1.588235
System Response =14.705380
Controller Response =1.705682
Pot_Speed =-1.627451
System Response =14.707160
Controller Response =1.707282
Pot_Speed =-1.588235
System Response =14.694200
Controller Response =1.708861
Pot_Speed =-1.588235
System Response =14.705660
Controller Response =1.710430
Pot_Speed =-1.666667
System Response =14.705920
Controller Response =1.712064
Pot_Speed =-1.549020
System Response =14.683010
Controller Response =1.713619
Pot_Speed =-1.509804
System Response =14.717980

```

7.0 Source Code Analysis

7.1 Main.C :

This file contains project initialisation information. The initialization is mainly achieved by the `project_Init ()` function, which completes the following task:

- Set special function register SYSCON register to 0x20. This means that the CAN baud rate prescaler is enabled. Ports 0, 2 and the signals `!RD` and `!WR` are not activated during the access to the XRAM/CAN controller. ALE generation is enabled.
- Set Special function IRCON to 0, which resets the register to clear all interrupt requests to prevent unintentional generation of an interrupt.
- Set EA to 1 to globally enable interrupts.
- Call `T01_vInit ()` function in file T01.C to initialize the Timer T0. It affects all necessary configurations of the SFR, depending on the selected operating mode. The configuration determines whether the T0 interrupts are to be released and the priority of the released interrupt.
- Call `ADC_vInit ()` function in ADC.C to initialize the Analog to Digital Converter. It effects all necessary configurations of the SFR, depending on the selected operating mode. The configuration determines whether the ADC interrupt is to be released and the priority of the released interrupt.
- Call `USART_vInit ()` function in the USART.C to initialize the USART.
- Call `CAN_vInit ()` function in CAN.C to initialize the CAN module.

7.2 T01.C File

Since the timer T0 is used for the sampling time, we define a variable as a counter to count the number of timer overflows so that we achieve a sampling interval of 0.1 second.

7.3 ADC.C File

We use the A/D converter in the continuous conversion mode for analog inputs from channel 0. A variable is defined to read the result of the A/D converter through the function `ADC_ubReadConv ()` in the ISR of the A/D converter. And then the same variable is used to change the result of the conversion through the calculation $x = x/255*5$ so that we can get the measured voltage. We use the **printf** command to send out the results via USART to the Hyper Terminal. The above-mentioned output is measured in float format to read decimal numbers with 2 decimal places.

7.4 USART. C

The USART is used for enabling the serial data transmission. As a result, we were able to view our results on the PC with the aid of the HyperTerminal program.

7.5 CAN.C

We initialised CAN so that a two-way communication between the motor model and the controller was obtained. Message objects 5 and 10 were used for this purpose, and their respective IDs were chosen to be 0x333 and 0x111. We decided to use 11 bit identifiers instead of the extended 29 bit identifiers. The baud rate was set to 100 baud.

8.0 Conclusion

Part 1 of the project, which had both the motor model and the PID controller on one microprocessor worked well. The controller was tested loading the codes in a computer and the motor model in another computer. The main idea was to emulate the local remote operation and the whole process was tested effectively. In addition to the above, every possible peripheral of the C505C was used in our project.

9.0 References

1. PID Controller Using the TMS320C31 DSK for Real-Time DC Motor Speed and Position Control , Jianxin Tang, Division of Electrical Engineering, Rulph Chassaing
2. C505C Keil Manual
3. Bosch Manual
4. Lecture Notes
5. Digital Signal Processing, Laboratory Experiments Using 'C' and the TMS320C31 DSK, Wiley. 1999
6. Basic DC Motor Speed Control With The Infineon C167 Family , <http://www.hitex.co.uk/c166/pidex.html>

Appendix I

```
%=====
% Simulated code for the DC Motor Operation
%=====

numg=22;
deng=[1 4];
numf=[0.167];
denf=[1];
T=0.1;
[numz,denz]=c2dm(numg,deng,T,...
'zoh');
numd=[1.001 -0.996];
dend=[1 -1];
[numz,denz]=series(numd,dend,...
numz,denz);
[numzc,denzc]=feedback(numz,...
denz,numf,denf);
[nums]=[2.5];
[dens]=[1];
[numc,denc]=series(nums,dens,...
numzc,denzc);
c=dstep(numc,denc,10001);
t=0:0.0005:5;
%dos('dsk3a control2');
%dos('dsk3load control2 BOOT');
plot(t,c),grid
xlabel('Time (seconds)')
ylabel('System output (rps)')
title('Time reponse')
```

Appendix II

'C' codes for the PID controller and the DC Motor model:

1. PID Controller processor codes:

a. Main.C

```
// USER CODE BEGIN (Main,0)
// Global variables

#define SlaveNum 2

data float V_ref = 0;
idata float Error[4];
data int int_temp = 0;
data float float_temp = 0;
idata float PID_Num[5] = {1.001,-0.996,0,0,0};
idata float PID_Den[5] = {1,-1,0,0,0};
idata float Controller_Output[4];
extern idata int fSystem_Output;
extern idata long New_Output = 0;

// USER CODE END

// The Main loop starts here

void main(void)
{
  // USER CODE BEGIN (Main,1)
  extern data ubyte count;
  // USER CODE END

  Project_Init();

  // USER CODE BEGIN (Main,2)

  // Initialize controller output to zero

  while (int_temp < 4)
  { Error[int_temp] = 0;
    Controller_Output[int_temp] = 0;
    int_temp++;
  }

  while (1)
```

```

{ Error[0] = V_ref - fSystem_Output;

//Modifying error array___setting previous vals

for(int_temp = 4; int_temp >= 1; int_temp --)
    { Error[int_temp] = Error[int_temp-1];
    }
if(count >= 3)
    { //PID Controller
    float_temp = 0;
    for(int_temp = 1; int_temp <= 4; int_temp ++ )
        { float_temp = ( float_temp - (
        PID_Den[int_temp]*Controller_Output[int_temp]));
        }
    for(int_temp = 0; int_temp <= 4; int_temp ++ )
        { float_temp = ( float_temp + ( PID_Num[int_temp]*Error[int_temp]));
        }
    Controller_Output[0] = float_temp;

// Modifying controller output array___Setting previous values

for(int_temp = 4; int_temp >= 1; int_temp --)
    { Controller_Output[int_temp] = Controller_Output[int_temp-1];
    }

New_Output = (long)(Controller_Output[0] + 15);
New_Output = New_Output*100; /*scale controller output in order retain two
                               decimal places */
CAN_vTransmit(10); /* Transmit controller output to dc motor model */

    count = 0;
    }
}
}
// USER CODE END

```

b. CAN.C

```

// USER CODE BEGIN (CAN_General,1)

#include "stdio.h"

#define SlaveNum 2
#define ID1 0x333 /* ID of processor that contains the DC motor model */
#define ID2 0x222 /* ID of processor that contains the PID controller */

// Define global variables

```

```

idata long SysOut = 0;
idata long ptrPot_Speed = 0;
idata long temp1 = 0;
idata long temp2 = 0;
idata long temp3 = 0;
idata long temp4 = 0;
idata float fSysOut = 0;
idata float fSystem_Output = 0;
idata ubyte tempByte1;
extern idata long New_Output;
idata float fNew_Output = 0;

// USER CODE END

// Receive Interrupt service routine for Message Object #5

// USER CODE BEGIN (CAN_IsrRxOk,5)
//*****

if (CAN_bNewData(5))
{ CAN_vReleaseObj( 5 );
}

CAN_vGetMsgObj(5, &RecvMsg);

if (RecvMsg.ulArbitr == ID1)           /* Wait for ID1 match */
{ temp1 = (long)RecvMsg.ubData[0];     /* Receive first byte of data */
  temp2 = (long)RecvMsg.ubData[1];     /* Receive second byte of data */

  temp2 = temp2 << 8;
  SysOut = temp1 | temp2;
  fSysOut = (float)(SysOut);
  fSysOut = fSysOut/100;               /* New speed of DC motor received */
  printf("System Response= %f\n",fSysOut);

  temp3 = (long)RecvMsg.ubData[2];     /* Receive third byte of data */
  temp4 = (long)RecvMsg.ubData[3];     /* Receive fourth byte of data */
  temp4 = temp4 << 8;
  ptrPot_Speed = temp3 | temp4;
  fSystem_Output = (float)(ptrPot_Speed);
  fSystem_Output = fSystem_Output/100;
  fSystem_Output = fSystem_Output - 15;
  printf("System_Output = %f\n",fSystem_Output);
}
// USER CODE END

```

```

// Transmit Interrupt service routine for Message Object #10

// USER CODE BEGIN (CAN_IsrTxOk,10)
//*****

tempByte1 = (ubyte)(New_Output & 0x000000FF);
CAN_OBJ[9].Data[0] = tempByte1;          /* First data byte of Controller output
to be transmitted */

tempByte1 = (ubyte)((New_Output & 0x0000FF00) >> 8);
CAN_OBJ[9].Data[1] = tempByte1;        /* Second data byte of Controller to be
transmitted */

fNew_Output = (float)New_Output;
fNew_Output = (fNew_Output/100);
fNew_Output = (fNew_Output-15);
printf("Controller Response = %f\n", fNew_Output); /* Display new controller output */

//*****
// USER CODE END

```

2. DC M/C Model and System Output Codes :

a. ADC.C

```

// USER CODE BEGIN (ADC_General,1)

#include <stdio.h>

// Begin Data declaration - GLOBAL

extern data ubyte   time;          // Count until 4 for 0.1 sec interval.

extern idata long   SysOut = 0;
extern idata long   ptrPot_Speed = 0;
extern idata float  ptrPot_Volt = 0;
extern idata float  fNew_Output;

// PID controller output

idata float  Controller_Output[4];

// DC Motor Transfer function
// DC motor discrete transfer function Numerator

idata float  DC_Num[5] = {0, 1.8151, -1.8060, 0, 0};

// DC motor discrete transfer function Denominator

```

```

idata float  DC_Den[5] = {1, -1.6703, 0.6703, 0, 0};
idata float  System_Output[4];

data unsigned ADC = 0;           // Value returned by AD converter
data float   Pot_Volt = 0;      // Voltage disturbance (Potentiometer)
data float   Pot_Speed = 0;     // Speed difference due to the voltage

// Temporary variable
data int     int_temp = 0;
data float   float_temp = 0;

// USER CODE END

// USER CODE BEGIN (ADC_Init,1)

// Initialising error, controller output, system output to zeroes

while(int_temp < 4)
{ Controller_Output[int_temp] = 0;
  System_Output[int_temp] = 0;
  int_temp++;
}

// USER CODE END

// USER CODE BEGIN (ADC_Isr,0)

// Reading AD value after every 0.1sec, Sampling Time
if (time == 5)
  { // Reading AD value

      (unsigned)ADC = ADC_ubReadConv();

// Voltage due to potentiometer (Range: 0V - +5V)

      Pot_Volt = ((float)ADC * 5)/ 255;
      ptrPot_Volt = Pot_Volt;

      Pot_Speed = (2 * Pot_Volt) - 5;

      Controller_Output[0] = fNew_Output;

      printf("Controller Response =%f \n", Controller_Output[0]);

// Modifying Controller Output array - setting previous values
for(int_temp = 4; int_temp >= 1; int_temp--)

```

```

{ Controller_Output[int_temp] = Controller_Output[int_temp - 1];
}

// Calculating current speed of motor

float_temp = 0;
for(int_temp = 1; int_temp <= 4; int_temp++)
{ float_temp = (float_temp - (DC_Den[int_temp] *
  System_Output[int_temp]) + (DC_Num[int_temp] *
  Controller_Output[int_temp]));
}
System_Output[0] = ((float_temp + 15 + Pot_Speed) * 0.166666666667)-
  2.5;

SysOut      = (long)((float_temp + 15)*100);

printf("System Response =%f\n", (float_temp + 15)); /* New speed of
  motor */

// Modifying System Output array - setting previous values

for(int_temp = 4; int_temp >= 1; int_temp--)

{ System_Output[int_temp] = System_Output[int_temp - 1];
}

ptrPot_Speed = (long)(System_Output[0]+15);
ptrPot_Speed = ptrPot_Speed*100;
printf("System Output =%f\n", System_Output[0]);

time = 0;
}

```

// USER CODE END

b. CAN.C

// USER CODE BEGIN (CAN_General,2)

// Define global variables

```

extern idata long SysOut ;
extern idata long ptrPot_Speed ;
idata long errorr=0;
idata float fNew_Output=0;
idata ubyte tempByte;
idata long temp1 =0;

```

```

idata long temp2 =0;
#define ID1 0x333
#define ID2 0x222

// USER CODE END

// Transmit Interrupt service routine for Message Object #5

// USER CODE BEGIN (CAN_IsrTxOk,5)
//*****

tempByte = (ubyte)( SysOut & 0x000000FF );
CAN_OBJ[4].Data[0] = tempByte; // set data byte 0

tempByte = (ubyte)(( SysOut & 0x0000FF00 )>>8);
CAN_OBJ[4].Data[1] = tempByte; //set data byte 1

tempByte = (ubyte)( ptrPot_Speed & 0x000000FF );
CAN_OBJ[4].Data[2] = tempByte;

tempByte = (ubyte)(( ptrPot_Speed & 0x0000FF00 )>>8);
CAN_OBJ[4].Data[3] = tempByte; //set data byte 1

//*****
// USER CODE END

// USER CODE BEGIN (CAN_IsrRxOk,10)
//*****

if ( CAN_bNewData(10) )
{ CAN_vReleaseObj(10) ;
}

CAN_vGetMsgObj(10, &RecvMsg);

if (RecvMsg.ulArbitr == ID2)
{ temp1 = (long)RecvMsg.ubData[0];
temp2 = (long)RecvMsg.ubData[1];
temp2 = temp2 << 8;
errorr = temp1 | temp2;
fNew_Output = (float)(errorr);
fNew_Output = (fNew_Output/100);
fNew_Output = fNew_Output - 15;
}

//*****
// USER CODE END

```

Appendix III

Attachments

A floppy disk containing the following files is attached.

1. Part '1'

This folder refers to Part 'A' of our project wherein both the motor model and the PID controller are loaded on one microprocessor and the results are displayed on a remote processor (uni-directional CAN).

a. Slave

This folder contains *.dav, *.c and *.Uv2 files for the PID controller + DC motor model equations.

b. Master

This folder contains *.dav, *.c and *.Uv2 files for the display program used to display the relevant data in a remote processor.

2. Part '2'

This folder refers to Part 'A' of our project wherein the motor model and the PID controller are loaded onto two separate microprocessors (bi-directional CAN).

a. Slave

This folder contains *.dav, *.c and *.Uv2 files for the DC motor model equations.

b. Master

This folder contains *.dav, *.c and *.Uv2 files for the PID Controller.