Session 2220

Development of a Matlab-Based Graphical User Interface Environment for PIC Microcontroller Projects

Sang-Hoon Lee, Yan-Fang Li, and Vikram Kapila

Department of Mechanical, Aerospace, and Manufacturing Engineering
Polytechnic University, Brooklyn, NY 11201
Email: [slee05@utopia, yli14@utopia, vkapila@duke].poly.edu

Abstract

Peripheral Interface Controllers (PICs) are inexpensive microcontroller units with built-in serial communication functionality. Similarly, Matlab, a widely used technical computing software, allows serial communication with external devices. In addition, Matlab provides graphical design tools such as Simulink and Dials and Gauges Blockset. This paper exploits the serial communication capability of PIC microcontrollers and the Matlab software along with graphical design tools of Matlab to create a Matlab-based graphical user interface (GUI) environment for PIC microcontroller projects. Three examples are included to illustrate that the integration of low-cost PIC microcontrollers with the Matlab-based GUI environment allows data acquisition, data processing, data visualization, and control.

1. Introduction

Peripheral Interface Controllers (PICs), developed and marketed by Microchip Technology, Inc. [1], are inexpensive microcontroller units that include a central processing unit and peripherals such as memory, timers, and input/output (I/O) functions on an integrated circuit (IC). There are more than 100 varieties of PIC microcontrollers available, each providing functionality for different types of applications [2], making PICs one of the most popular microcontrollers for educational, hobby, and industrial applications. Similar to other microcontrollers, PICs are usually not designed to interface with human beings; instead they are directly embedded into automated products/processes. Thus, graphical user interface (GUI) capabilities, which have become a mainstay of many personal computer (PC) applications, are nonexistent for PICs.

Endowing PIC-based projects with GUI tools can speed the development process in data driven applications such as feedback control, smart sensors, etc. Microchip Technology's emulator and debugger products (e.g., MPLAB IDE, MPLAB-ICE) are very helpful in debugging PIC source code and emulating user-written programs. However, these tools do not provide data co-processing and advanced data visualization capabilities.

Fortunately, PIC microcontrollers include serial communication functionality to facilitate data communication with external devices such as analog-to-digital converters (ADC), 1-wire sensors, etc. Similarly, Matlab, a commercially available interactive mathematical programming software, also provides serial data communication functionality on PCs. In addition, Simulink, Matlab's interactive icon-based programming environment, enables users to simulate and analyze dynamic system models. Finally, the Dials and Gauges Blockset of Simulink allows users to embed control objects (e.g., sliders, knobs) and display objects (e.g., graphs, gauges) in Simulink models to develop an interactive GUI environment. In this paper, we exploit the serial communication functionality of Matlab to enable a PC to communicate with PIC microcontrollers to transmit control commands and receive sensory data. In addition, we utilize Matlab, Simulink, and Dials and Gauges Blockset to develop an interactive GUI environment for PIC projects, allowing enhanced data processing and visualization.

In this paper, we use a PIC16F74, 40-pin, 8-bit CMOS FLASH dual inline package IC. To facilitate serial communication between PIC and PC, we interface a RS232 driver/receiver with the PIC16F74. The effectiveness of our Matlab-based GUI environment to interact with PIC microcontroller projects is demonstrated by using three examples: (1) export user commands from a Simulink GUI to an actuator interfaced to the PIC, (2) import signals from a sensor interfaced to the PIC into a Simulink GUI, and (3) use Simulink GUI to export user commands to the PIC and import sensory data from the PIC to control a device and monitor its status.

2. Hardware Environment

The hardware environment for this paper consists of a PIC microcontroller, a PC, a RS232 driver/receiver, and a DB-9 serial cable. The PIC microcontroller is interfaced with external devices such as sensors (e.g., photoresistors) and actuators (e.g., servomotors). In addition, the PIC microcontroller performs embedded computing. The PC is used to write user specified embedded programs to be executed by the PIC microcontroller. Furthermore, the PC hosts an interactive GUI for the user to manipulate control variables and visualize sensory data. The PIC microcontroller and the PC communicate using a serial interface. A PIC development board (see section 2.4) and a light refraction experiment test bed (see section 2.5) are used to illustrate our PIC-based data acquisition and control approach.

2.1. Peripheral Interface Controller

PIC microcontrollers are small, low-cost controllers that include a processor and a variety of peripherals. PICs are significantly easier to use *vis-à-vis* embedded microprocessors. As an example, users can assign desired functionality (e.g., ADC, USART[1]) to I/O pins of PICs. PICs can be operated at various clock speeds (32 kHz to 20 MHz). PIC's memory architecture separates its data memory from its program memory with the program memory available as One-Time Programmable (OTP), Erasable Programmable Read-Only Memory (EPROM), or FLASH. PICs are programmed in the PIC assembly language using a 35 single-word instruction set. See [3] for more details on hardware and software features of PIC microcontrollers.

The user specified embedded PIC program is written on the PC and downloaded from the PC to the PIC microcontroller using the DB-9 serial cable connection between the PC and a PIC Development Programmer on which the PIC microcontroller is installed. Commonly available PIC Development Programmers include PICSTART Plus [4] from Microchip, Inc., and PIC-PG2B, a handy, low-cost programmer [5] from Olimex Ltd., among others. In this paper, we use the PICSTART Plus programmer that requires MPLAB Integrated Development Environment, a free software available on the Microchip website, for programming PICs.

In this paper, we employ a PIC16F74, a 40-pin CMOS FLASH-based, 8-bit, mid-range (14-bit instruction word length) microcontroller (see Figure 1). PIC16F74 has 4 Kbytes of FLASH program memory and 192 bytes of data memory. Furthermore, it has 33 digital I/O pins organized in 5 groups of I/O ports that can be assigned as 8-bit ADC, Capture/Compare/PWM[2] (CCP), the 3-wire Serial Peripheral Interface (SPI), the 2-wire Inter-Integrated Circuit (I^2C) bus, USART ports, etc. We use an external 20 MHz high-speed crystal oscillator to supply operating clock cycles. The PIC16F74 can be powered using a wide range of voltage sources, e.g., 2-volt direct current (VDC) to 5.5VDC, and each of its I/O pin can sink or source up to 25mA of current. It is ideal not only for laboratory data acquisition (the application considered in this paper), but also for automotive, industrial, and consumer applications.

(a) (b)

Figure 1: (a) PIC16F74 (b) Pin diagram of PIC16F74

2.2. Personal Computer

In this paper, an IBM-compatible Pentium 3 PC running Microsoft Windows NT 4.0 operating system is used. As previously mentioned, the PC is used to write, debug, and download

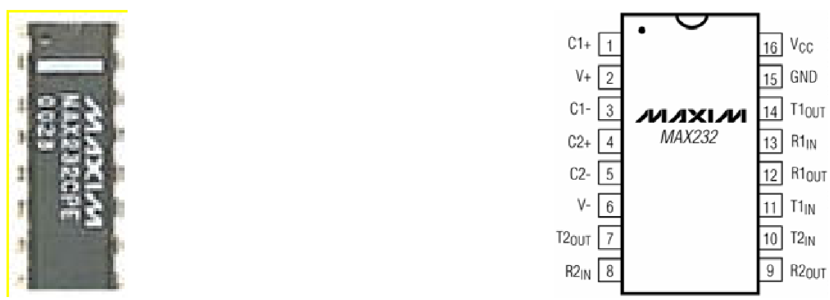
---

[1] Universal synchronous/asynchronous receiver and transmitter.
[2] Pulse width modulation.

embedded PIC programs. One of the serial ports on the PC is reserved for serial communication with the PIC microcontroller. MPLAB, Matlab (version 6.1), Simulink, and Dials and Gauges Blockset are installed on the PC. Control variables are manipulated via the PC by interacting with control panels embedded in the Simulink program. In addition, all experimental data is collected and displayed on the PC in display panels embedded in the Simulink program.

2.3. RS232 Driver/Receiver

MAX232 (see Figure 2) is a 2-channel, RS232 driver and receiver manufactured by Maxim Integrated Products, Inc. It requires a 5VDC power supply and converts voltage levels between PC-based logic and PIC microcontroller-based logic. Specifically, whereas the voltage levels of logic high and logic low for the PC correspond to –12VDC and 12VDC, respectively, like many other microcontrollers the logic high and low for the PICs correspond to 5VDC and 0VDC, respectively. The MAX232 is used with five 1μF capacitors to adjust the voltage level differences between the PC-based logic and the PIC-based logic. See [6] for more details of the MAX232 hardware features.

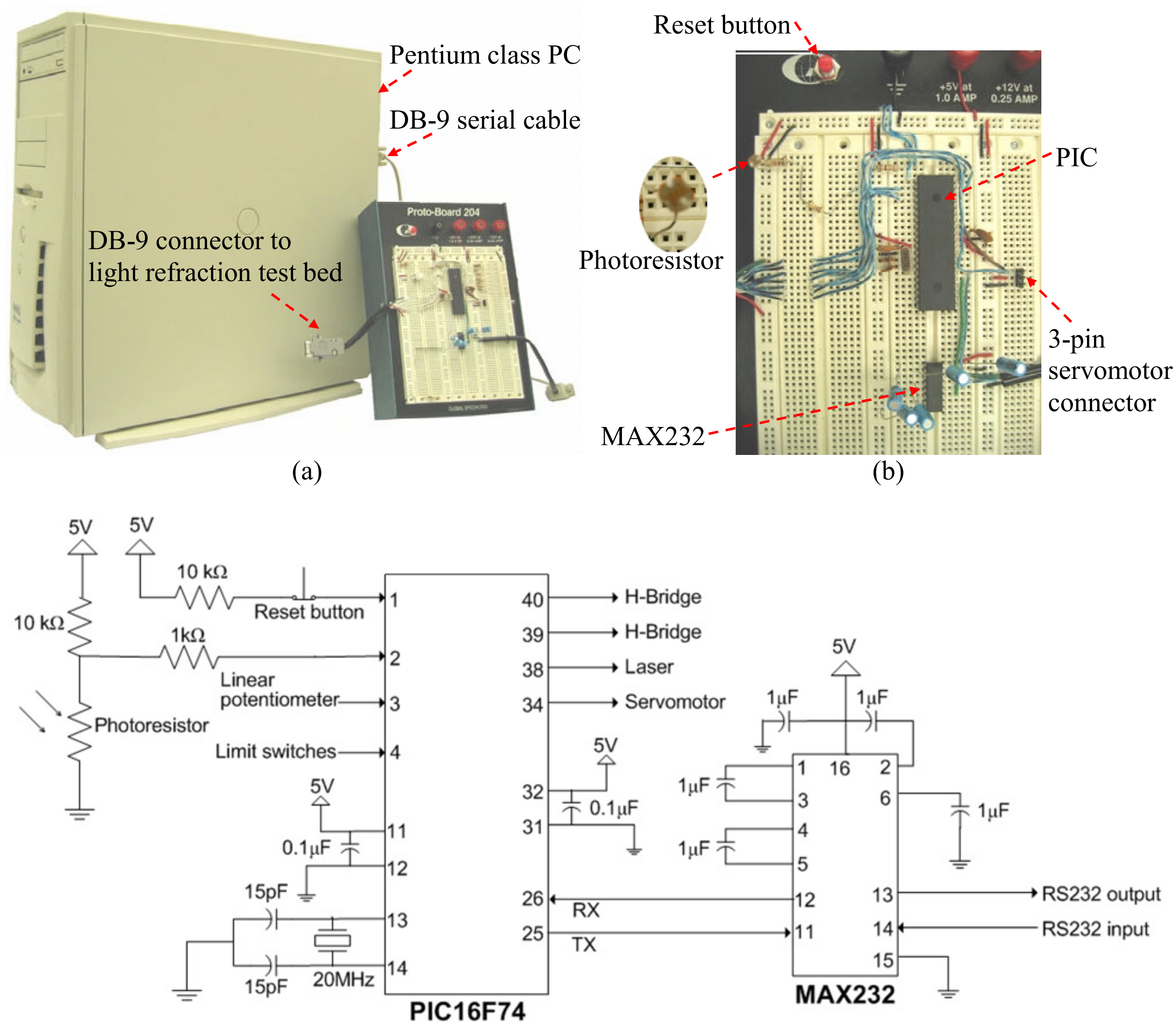


(a) (b)

Figure 2: (a) MAX232 (b) Pin diagram of MAX232

2.4. PIC Development Board

The PIC development board (see Figure 3) consists of a sensor (photoresistor), a 3-pin header for a servomotor connection, a 20MHz crystal oscillator, a MAX232 with five 1μF capacitors, a PIC16F74 microcontroller, a breadboard, and two DB-9 connectors. The photoresistor sensor provides light intensity measurement and is interfaced to a pin allocated as an 8-bit ADC in port A of the PIC16F74 microcontroller. The circuit diagram of Figure 3(c) illustrates how various sensors and actuators of the light refraction experiment test bed (see section 2.5) are interfaced to the PIC microcontroller. The PIC transmits/receives sensory data to/from the PC via the MAX232. A red reset button is connected to the Master Clear (MCLR) pin of the microcontroller.

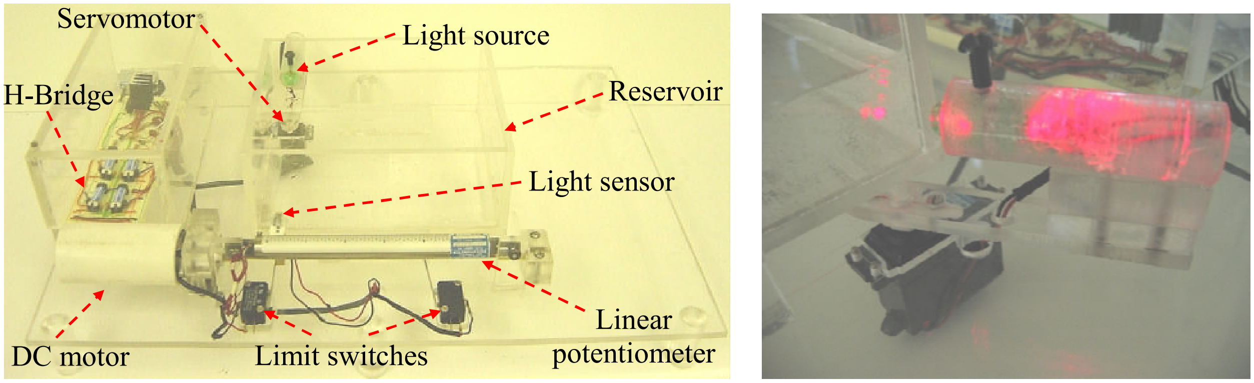


(c)

Figure 3: (a) PC and PIC development board (b) Larger view of the PIC development board
(c) Circuit diagram of the PIC development board

2.5. Light Refraction Test Bed

The light refraction test bed (see Figure 4) is a mechatronics-aided physics experiment developed under a National Science Foundation (NSF) sponsored Science and Mechatronics Aided Research for Teachers (SMART) program [7] at Polytechnic University. This experiment is designed to demonstrate the law of light refraction. It consists of a light source, a light sensor, a linear potentiometer, two limit switches, a servomotor, a DC motor, a liquid reservoir, and necessary circuitry. A liquid reservoir on the top of the test bed can store various liquid media whose index of refraction needs to be determined. For simplicity, in this paper, we use water from a water fountain as the test liquid.



(a) (b)

Figure 4: (a) Light refraction experiment test bed (b) Light source mounted on the servomotor

On one side of the tank, as shown in Figure 4(b), a laser pointer, used as the light source, is mounted on the arm of the servomotor that sets the angular position of the light source to the incidence angle specified by the user. On the other side of the tank, a general Cadmium Sulfide (CdS) photoresistor, used as the light sensor, is mounted on the wiper of the linear potentiometer. It monitors the refracted light coming out from the liquid reservoir (see Figure 5). A DC motor drives the light sensor along the linear potentiometer by turning a motor shaft connected to a brass screw rod thereby transforming rotary motion into linear motion. Limit switches at each end of the linear potentiometer indicate sensor travel limit. The photoresistor and the linear potentiometer output analog voltage signals between 0VDC and 5VDC.

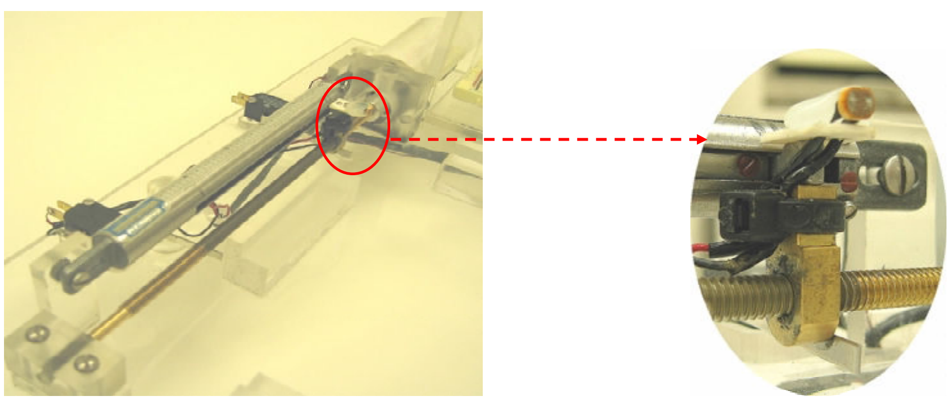


Figure 5: Detailed view of the light sensor traveling along the linear potentiometer

3. Software Environment

The software environment for this paper consists of the PIC assembly language, Matlab, Simulink, and Dials and Gauges Blockset. The PIC assembly language is a primitive programming language consisting of a 35 single-word instruction set. Matlab is an interactive technical computing software. Simulink is Matlab's model-based, system-level, visual programming environment that is widely used to simulate and analyze dynamic system models using icon-based tools. Finally, the Dials and Gauges Blockset of Simulink provides an ability to embed visual, realistic-looking, virtual instrumentations in Simulink models. In this paper, these software tools are judiciously synthesized to produce an effective, interactive GUI environment. In the sequel, we summarize key instructions of the PIC assembly language and Matlab that enable serial communication between PIC microcontroller and Matlab GUI running on the PC.

3.1. PIC Assembly Program

As indicated above, the PIC assembly language consists of a 35 single-word instruction set (see datasheets [8] for details). The PIC data memory is partitioned into several banks (e.g., 5 banks for PIC16F74) that contain the general-purpose registers and the special-function registers. The special-function registers are used to set up special operations (e.g., ADC, USART, and PWM) and to watch the status of the special operations (e.g., the availability of transmission or reception of the USART). Below, we review key PIC instructions and special function registers used for serial communication functionality.

3.1.1. Key PIC instructions

BCF: Bit clear f
Syntax: [label] *BCF f, b*
BCF literally means that the b^{th} bit in the register 'f' is cleared. BCF sets the b^{th} bit in the register 'f' to zero, logic low.

BSF: Bit set f
Syntax: [label] BSF f, b
BSF instruction does the opposite of BCF, i.e., it sets the b^{th} bit in the register 'f' to one, logic high.

MOVLW: Move literal to w
Syntax: [label] MOVLW k
The literal 'k' is loaded into the working register. The literal 'k' can be expressed in terms of an 8-bit binary, decimal, or hexadecimal number. For example, b'00101111' in 8-bit binary is equivalent to 0x2F in hexadecimal. Note that the prefixes b, 0x, and d declare the data type to be binary, hexadecimal, and decimal, respectively.

MOVWF: Move w to f
Syntax: [label] MOVWF f
MOVWF transfers data from the working register to the specified register 'f.' Since the literal 'k' cannot be directly assigned into the specified register 'f,' the literal 'k' is first assigned to the working register (e.g., MOVLW k) and then moved into the register 'f' (e.g., MOVWF f).

BTFSS: Bit test f, skip if set
Syntax: [label] BTFSS f, b

BTFSS checks the b^th^ bit in the specified register 'f,' and executes the next instruction if this bit is zero. Alternatively, if the bit is one, the next instruction is skipped, and the following instruction is executed.

3.1.2. *Special function registers used for serial communication functionality*

MOVLW d'value'
MOVWF SPBRG
The special function register 'SPBRG' contains the user-specified baud rate for serial communication. In particular, the command MOVLW d'129' places 129 in the working register. Next, the command MOVWF SPBRG moves the content of the working register to the special function register 'SPBRG.' The placement of 'value' 129 in the 'SPBRG' register sets the baud rate to 9,600.

MOVLW b'clock source select bit, 9-bit transmit enable bit, transmit enable bit, usart mode select bit, unimplemented, high baud rate select bit, transmit shift register status bit, 9th bit of transmit data'
MOVWF TXSTA
The special function register 'TXSTA' contains information for the data-transmit status and control in an 8-bit binary expression. In particular, the use of commands MOVLW b'00100100' and MOVWF TXSTA, sets up the 'TXSTA' register to enable 8-bit, high speed asynchronous serial data transmission.

MOVLW b'serial port enable bit, 9-bit receive enable bit, single receive enable bit, continuous receive enable bit, unimplemented, framing error bit, overrun error bit, 9th bit of received data'
MOVWF RCSTA
The special function register 'RCSTA' contains information for the data-receive status and control in an 8-bit binary expression. In particular, the use of commands MOVLW b'10010000' and MOVWF RCSTA, sets up the 'RCSTA' register to enable 8-bit, continuous asynchronous serial data reception.

3.2. Matlab Program

Matlab is a commercially available, widely used, interactive, technical computing software. Matlab's versions 6.1 and higher provide serial communication functionality. To serially communicate with an external device from Matlab, the following steps need to be performed. First, create a serial port object to identify the specific serial port of the PC connected to the external device. In addition, specify how this serial port is to be configured (i.e., baud rate, number of data bits, etc.). Second, connect the serial port object created above to the external device. Third, send command signals to the external device and receive data from the external device. Fourth, disconnect serial communication connection from the external device and close the serial port object. Finally, release control of the serial port. Next, we list the key Matlab instructions used for serial communication. See [9] for further details.

serial (the PC serial port, the baud rate, the number of data bits)
This command is used to create a new serial port object. In addition, it configures the serial port properties. In this paper, we used the COM2 serial port of the PC with 9,600 baud rate.

fopen (object)
This command opens the serial port object just created and connects the PC to the external device for actual serial communication.

fread/fwrite (object, size, precision)
The 'fread' command enables the PC to read binary data from the external device. Alternatively, the 'fwrite' command enables the PC to send control data in binary format to the external device.

fclose (object)
This command closes the serial port object, thereby disconnecting serial communication between Matlab and the external device.

freeserial(port)
Once Matlab establishes a data link with the serial port, it assumes complete control of the serial port. The 'freeserial' command is used, after closing the port object using the 'fclose' command, to force Matlab to relinquish control of the serial port. The command takes on one argument, the port that was used for data communication. This command is executed from the Matlab command line after the termination of experiment.

3.2.1. Simulink

Simulink is Matlab's interactive, icon-based programming environment [10]. It enables users to build block diagrams to simulate and analyze dynamic system models. Designers can effortlessly transfer paper designs of dynamic systems into Simulink block diagrams. Simulink block diagrams can be modified as easily as paper models of dynamic systems. In addition, Simulink allows for detailed monitoring of dynamic system outputs at any point in the block diagram using various tools (e.g., Scope, Display, etc.). Finally, data processing tasks such as signal scaling, filtering, etc., can be easily performed in Simulink.

3.2.2. Dials and Gauges Blockset

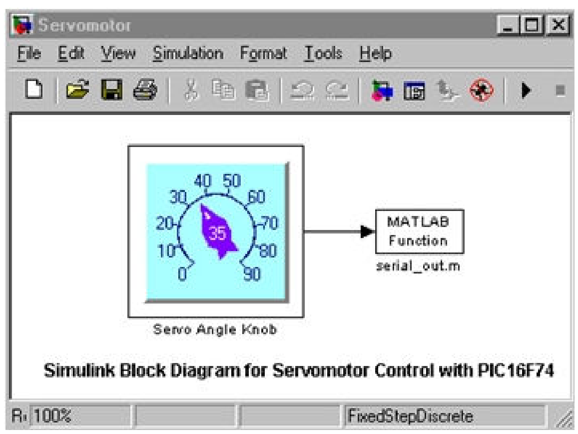
The Dials and Gauges Blockset [11] provides enriched views of graphical, 3-D instruments called virtual instruments. It has various templates that can be customized to create realistic virtual instruments for electrical, aerospace, automotive, medical, and process control systems. The virtual instruments created using the Dials and Gauges Blockset dynamically interact with Matlab and Simulink, thus providing an interactive interface for users to enter command inputs and visualize sensory outputs.

4. Examples of Serial Communication between PIC and PC

4.1. Serial Communication from PC to PIC: Servomotor Position Control

This example illustrates one-directional serial communication from the PC to the PIC microcontroller. In particular, it demonstrates that the user commands from a Simulink block diagram can be exported to an actuator interfaced to the PIC microcontroller. The example focuses on servomotor position control.

The Simulink block diagram for this example is shown in Figure 6. It consists of a dial, from the Dials and Gauges Blockset, denoted as the servo angle knob. The user interacts with the dial to enter servomotor position control command. The dial has a range from 0 to 90 degrees with one-degree resolution. The value of the angle commanded by the user is shown in the middle of the knob. The Matlab m-function block next to the knob contains a Matlab m-file to perform serial communication from the PC to the PIC. The user specified servomotor position control command is transmitted to the PIC via a serial cable connection between the PC and the PIC. When the PIC receives the command angle, it assigns the angle to a variable in the PIC code. Next, the PIC utilizes the command angle to compute, generate, and apply pulse trains for servomotor position control. In this example, we used a 6VDC standard servomotor that is interfaced to the 3-pin servomotor connection header on the PIC development board (see Figure 3). The PIC assembly code corresponding to this example is available in Appendix A.




```
%Matlab function serial_out.m for serial communication from PC to PIC
function serial_out(angle)                    %serial_out function defined
ser_obj=serial('COM2','baudrate',9600);       %create and configure a serial port object
fopen(ser_obj);                               %connect the serial port object to the device
ServoCommand=round(angle+107.3);              %input for servomotor where 107.3 refers to offset
fwrite(ser_obj,[ServoCommand],'async');       %send user command, i.e., dial input, to the PIC
pause(1);
fclose(ser_obj);                              %disconnect the serial port object from the device
```

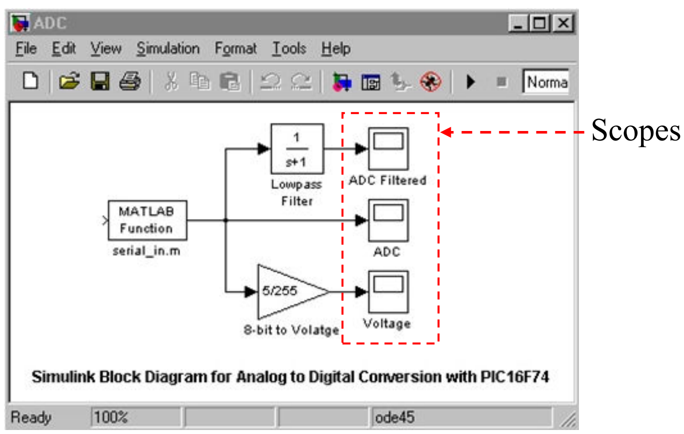
Figure 6: Simulink block diagram and m-function for PC to PIC serial communication

4.2. Serial Communication from PIC to PC: Data Acquisition, Processing, and Plotting

This example illustrates one-directional serial communication from the PIC microcontroller to the PC. In particular, it demonstrates that a Simulink block diagram can be designed to acquire measurement from a sensor that is interfaced to the PIC. The example focuses on acquiring measurements from a photoresistor that senses light intensity.

Referring to Figure 3 (c), a light sensor is constructed by connecting a 10 KΩ resistor and a photoresistor in a voltage divider circuit. The output of the light sensor varies depending on the light intensity incident upon the photoresistor; here the light sensor output refers to the voltage at the junction of the 10 KΩ resistor and photoresistor. This output is connected to I/O pin 2 of the PIC16F74. The I/O pin 2 is configured as an ADC in the PIC assembly code. Each time, the PIC assembly code tasks the PIC to measure the light sensor output, the PIC16F74 converts the analog voltage signal at the voltage divider output into a corresponding 8-bit digital value. Thus, when the photoresistor is placed in dark condition, the 8-bit ADC returns a value close to 255. Alternatively, when the photoresistor is exposed to bright light, the ADC returns a value close to 0.

The Simulink block diagram for this example is shown in Figure 7, where a Matlab m-function is used to acquire the digitized output of the sensor using serial communication. The Simulink block diagram of Figure 7 also processes and plots the sensory data. In particular, the top scope in Figure 7 plots the light intensity measurement (in terms of digitized output of the

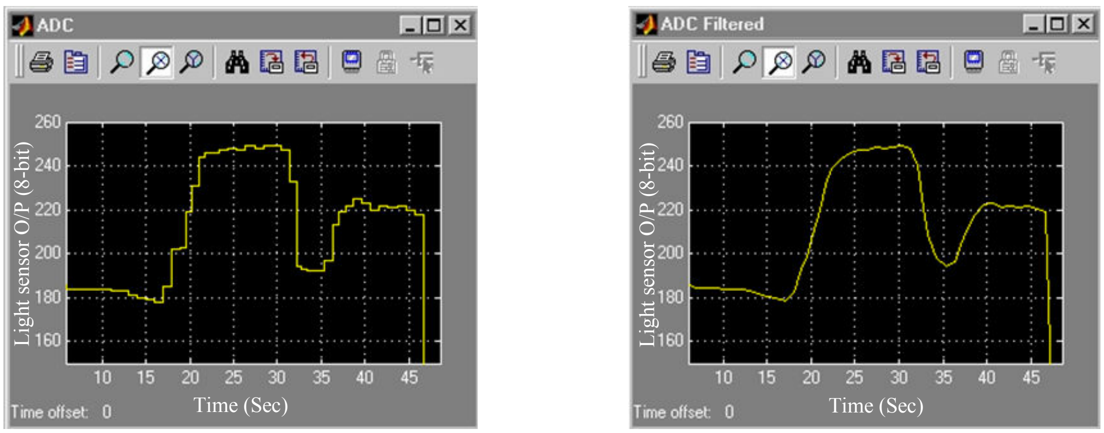



```
%Matlab function serial_in.m for serial communication from PIC to PC
function v=serial_in(dmyin)                  %serial_in function defined
ser_obj=serial('COM2','baudrate',9600);      %create and configure a serial port object
ser_obj.ReadAsyncMode = 'manual';            %specify an asynchronous read operation
fopen(ser_obj);                              %connect the serial port object to the device
LightSensOut=fread(ser_obj,1,'uint8');       %read the light sensor output
fclose(ser_obj);                             %disconnect the serial port object from the device
v=LightSensOut;                              %8-bit representation of the light sensor output
```

Figure 7: Simulink block diagram and m-function for PIC to PC serial communication

voltage divider circuit) versus time, where the measurements are filtered using a low-pass filter. The middle scope plots the unfiltered light intensity measurement. Finally, the bottom scope plots the light intensity measurement in terms of voltages by processing the 8-bit digital value of the voltage divider circuit through a gain factor.

An experiment was conducted in which the light intensity was abruptly altered at several time instances. The response plots acquired and processed using the Simulink block diagram of Figure 7 are shown in Figure 8. The filtered output response in Figure 8(b) is much smoother than the unfiltered response in Figure 8(a). Thus, Figure 8 demonstrates the efficacy of signal co-processing using Matlab and Simulink for PIC-based projects. The PIC assembly code corresponding to this example is available in Appendix B.



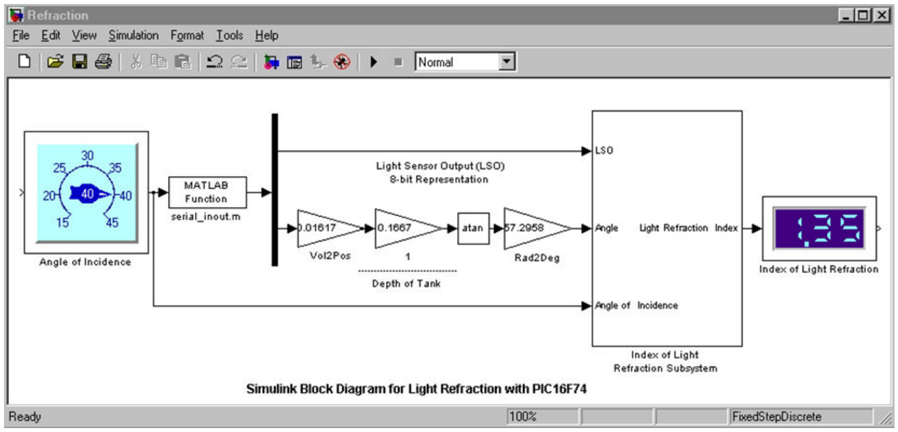
(a) (b)

Figure 8: (a) Unfiltered plot of ADC and (b) Filtered plot of ADC

4.3. Bi-directional Serial Communication between PIC and PC

In this example, the light refraction test bed is used to demonstrate the advantage of exploiting bi-directional serial communication between PIC and Matlab-based GUI executing on the PC. A Simulink-based interactive GUI for the light refraction test bed is shown in Figure 9. The user interacts with the dial object to command the angle of incidence of light source. The Matlab m-function block next to the knob contains a Matlab m-file that transmits the user command input to the PIC serially. The PIC stores the user input in a variable and uses it to compute, generate, and apply pulse trains to control servomotor position. This positions the light source, mounted on the servomotor arm, at the commanded angle of incidence. Next, the PIC turns on the light source and performs the following tasks: drive the light sensor along the linear potentiometer by turning the DC motor, measure the position of the light sensor along the linear potentiometer and the corresponding output of the light sensor, and transmit the position and light sensor measurements to the PC. The Matlab m-function block shown in Figure 9 enables receipt of the position and light sensor measurements from the PIC serially. Simulink blocks following the m-file function block are used for various data processing tasks, e.g., conversion of position measurement to the refraction angle and computation of index of light refraction.

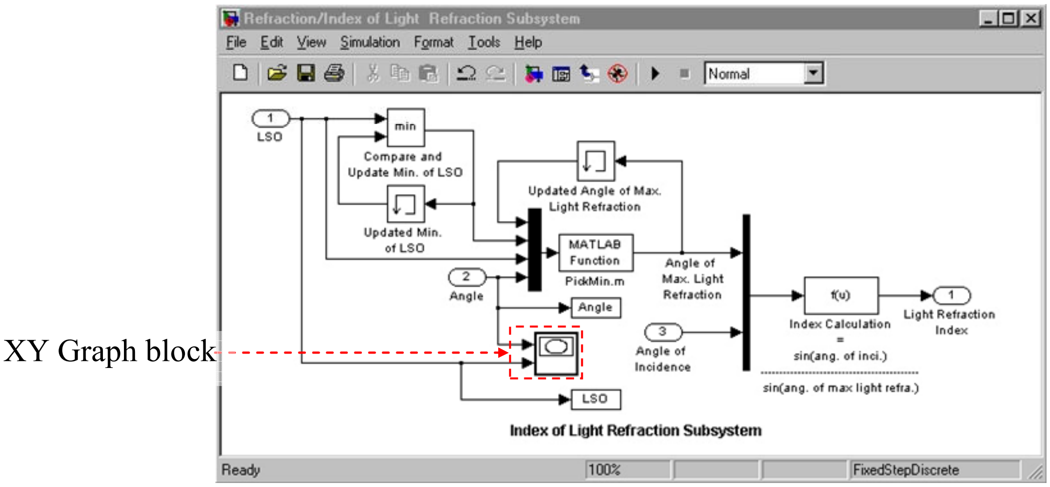
Finally, a generic numeric LED display, from the Dials and Gauges Blockset, is used to indicate the calculated value of index of light refraction for the experimental liquid.




```
%Matlab function serial_inout.m for bi-directional serial communication between PIC and PC
function V=serial_inout(angle)                %serial_inout function defined
ser_obj=serial('COM2','baudrate',9600);       %create and configure a serial port object
ser_obj.ReadAsyncMode = 'manual';             %specify an asynchronous read operation
fopen(ser_obj);                               %connect the serial port object to the device
ServoCommand =round(angle+107.3);             %input for the servomotor where 107.3 refers to offset
fwrite(ser_obj,[ServoCommand],'async');       %send user command, i.e., dial input, to the PIC
LightSensOut =fread(ser_obj,1,'uint8');       %read the light sensor output from the PIC
Position=fread(ser_obj,1,'uint8')+9;          %read the linear potentiometer output from the PIC
fclose(ser_obj);                              %disconnect the serial port object from the device
V=[LightSensOut;Position];                    %output in matrix form
```

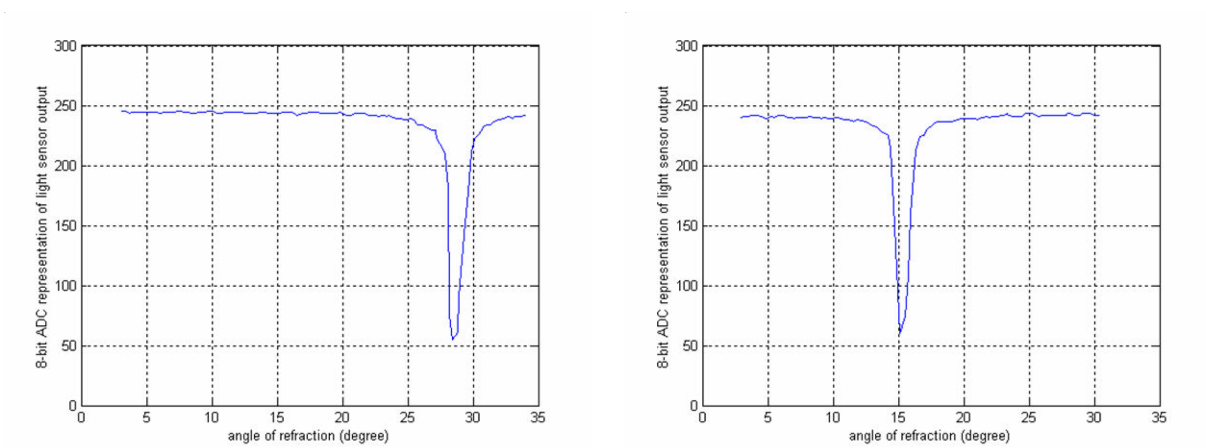
Figure 9: Simulink block diagram and m-function for bi-directional serial communication between PIC and PC

Figure 10 shows the block diagram of index of light refraction subsystem of Figure 9. The block diagram of Figure 10 is used to generate a plot of angle of refraction versus the light sensor output. Figure 11 shows the plots of angle of refraction versus the light sensor output for two commanded values of incidence angle, namely, 40˚ and 20˚. Note that for each incidence angle, the index of refraction is computed from the angle of refraction corresponding to the smallest output returned by the light sensor. Thus, the block diagram of Figure 10 is also used to calculate the index of light refraction. The Matlab m-function in this subsystem monitors and captures the angle data corresponding to the smallest measurement returned by the light sensor. Note that the light sensor output is smallest when the intensity of refracted light focused on the light sensor is highest. Next, the angle data is used to compute the index of light refraction. The PIC assembly code corresponding to this example is available in Appendix C.




```
%Matlab function PickMin.m for capturing the angle of refraction when max. light is on the light sensor
function y=PickMin(minangle,minLSO,LSO,angle)  %PickMin function defined
if LSO <= minLSO                                %condition loop for updating the minangle
   minangle=angle;                              %update minangle with respect to min. LSO
end
y=minangle;                                     %show the angle at the most light intensity
```

Figure 10: Simulink block diagram and m-function for calculating the index of light refraction



(a) (b)

Figure 11: Angle of refraction vs. light sensor output for incidence angle (a) 40˚ and (b) 20˚

5. Conclusion

In this paper, we developed and presented Matlab-based GUIs for PIC microcontroller projects by exploiting Simulink, Dials and Gauges Blockset, and serial communication capabilities of Matlab and PIC. Three examples were presented to illustrate the productivity enhancement potential of the Matlab-based GUI environment when developing PIC microcontroller projects. The GUIs designed using framework of this paper allow the user to:

vary control commands, acquire sensory data, perform on-line data processing, and visualize data using realistic looking virtual instruments. Note that the framework of this paper allows the use of microcontroller as a low-cost, stand-alone Data Acquisition and Control Board (DACB). Whereas PC-based DACBs typically cost several hundred to over thousand dollars, a PIC microcontroller costs only a few dollars. Thus, the use of PIC microcontrollers with the proposed Matlab-based GUI environment provides a low-cost DACB solution that can be particularly beneficial to educators.

Appendix

Appendix A. PIC Assembly Code for Serial Communication from PC to PIC

```
;This code is used to control angular position of a servomotor
;1. Receive user command from PC
;2. Generate pulse train to drive servomotor to a desired angle
 LIST p=16f74
 INCLUDE "p16f74.inc"
 __CONFIG _CP_OFF & _WDT_OFF & _HS_OSC &
_PWRTE_ON ;configure PIC16F74
counter EQU 20h ;file address of counter var
iteration EQU 21h ;file address of iteration var
tempval  EQU 22h ;file address of tempval var
 ORG 0 ;origin address is 0
 CLRF STATUS ;clear status register
 GOTO BootStart ;go to BootStart
BootStart
 BANKSEL PORTA ;select bank 0
 CLRF PORTB ;clear portB
 CLRF PORTC ;clear portC
 BANKSEL TRISA ;select bank 1
 MOVLW b'00000000'
 MOVWF TRISB ;set PORTB as all outputs
 MOVLW b'10000000'
 MOVWF TRISC ;set RC7 as input
TimerInitialization
 BSF  STATUS, RP0 ;select bank 1
 MOVLW b'00000001'
 MOVWF OPTION_REG ;set prescaler of TMR0 to 1:4
 BCF STATUS, RP0 ;select bank 0
 MOVLW b'10000100'
 MOVWF INTCON ;enable all unmasked interrupts
              ;and TMR0 register overflow
 CLRF TMR0 ;clear timer
BaudRateSettingsforUSART
 BSF STATUS, RP0 ;select bank 1
 MOVLW d'129'
 MOVWF SPBRG ;set baudrate 9600 for 20MHz crystal
 MOVLW b'00100100'
 MOVWF TXSTA ;8-bit asyn. high-speed transmission
 BANKSEL RCSTA ;select bank 0
 MOVLW b'10010000'
 MOVWF RCSTA ;8-bit asyn. continuous reception
 MOVF RCREG, W
 MOVF RCREG, W
 MOVF RCREG, W ;flush reception buffer 3 times
MainProgram
 BCF STATUS, RP0 ;select bank 0
Check BTFSS PIR1, RCIF ;check if data is received
 GOTO Check
 MOVF RCREG, W ;move received data to W
 MOVWF tempval ;save data from W into tempval
 MOVLW 0x64
 MOVWF iteration ;save iteration value for pulse train
BeginServo
 MOVF tempval, 0 ;move tempval to W
 MOVWF counter ;save data from W into counter
LoopHigh
 CLRF TMR0 ;clear timer
 BSF PORTB, 1 ;set RB1 to high
 MOVLW 0x05
InnerLoopHigh
 SUBWF TMR0, 0 ;set and countdown timer
 BTFSS STATUS, 2 ;check if timer is zero
 GOTO InnerLoopHigh ;go to InnerLoopHigh again
 BCF STATUS, 2 ;reset zero bit of status
 DECFSZ counter ;countdown counter and check if zero
 GOTO LoopHigh ;go to LoopHigh
 BCF PORTB, 1 ;set RB1 to low
 MOVLW 0xfa
 MOVWF counter ;set the value of counter for low
LoopLow
 CLRF TMR0 ;clear timer
 BCF STATUS, 2 ;reset zero bit of status
 MOVLW 0x15
InnerLoopLow
 SUBWF TMR0, 0 ;set and countdown timer
```

```
BTFSS STATUS, 2 ;check if timer is zero
GOTO InnerLoopLow ;go to InnerLoopLow again
BCF STATUS, 2 ;reset zero bit of status
DECFSZ counter ;countdown counter and check if zero
GOTO LoopLow ;go to LoopLow
BCF STATUS, 2 ;reset zero bit of status
DECFSZ iteration ;countdown iteration, check if zero
GOTO BeginServo ;go to BeginServo
GOTO MainProgram ;go to MainProgram to repeat
END ;end line of the code
```

Appendix B. PIC Assembly Code for Serial Communication from PIC to PC

```
;This code is used to collect the light sensor output
;1. Measure the voltage output from photoresistor
;2. Send the digitized output to PC using USART
 LIST p=16f74
 INCLUDE "p16f74.inc"
 __CONFIG _CP_OFF & _WDT_OFF & _HS_OSC &
_PWRTE_ON ;configure PIC16F74
 ORG 0 ;origin address is 0
 CLRF STATUS ;clear status register
 GOTO BootStart ;go to BootStart
BootStart
 BANKSEL PORTA ;select bank 0
 CLRF PORTA ;clear portA
 CLRF PORTC ;clear portC
 BANKSEL TRISA ;select bank 1
 MOVLW b'00000001'
 MOVWF TRISA ;set RA0 as input
 MOVLW b'10000000'
 MOVWF TRISC ;set RC7 as input
ADCInitialization
 BCF STATUS, RP0 ;select bank 0
 MOVLW B'10000001'
 MOVWF ADCON0 ;enable ADC and select CH0
 BSF STATUS, RP0 ;select bank 1
 MOVLW b'00000100'
 MOVWF ADCON1 ;set RA0, 1, and 3 to A/D ports
TimerInitialization
 BSF STATUS, RP0 ;select bank 1
 MOVLW b'00000001'
 MOVWF OPTION_REG ;set prescaler of TMR0 to 1:4
 BCF STATUS, RP0 ;select bank 0
 MOVLW b'10000100'
 MOVWF INTCON ;enable all unmasked interrupts
                  ;and TMR0 register overflow
 CLRF TMR0 ;clear timer
BaudRateSettingsforUSART
 BSF STATUS, RP0 ;select bank 1
 MOVLW d'129'
 MOVWF SPBRG ;set baudrate 9600 for 20MHz crystal
 MOVLW b'00100100'
 MOVWF TXSTA ;8-bit asyn. high-speed transmission
 BANKSEL RCSTA ;select bank 0
 MOVLW b'10010000'
 MOVWF RCSTA ;8-bit asyn. continuous reception
StartADCandUSART
 CALL ADCLight ;call ADCLight subroutine
 CALL Send ;call Send subroutine
 GOTO StartADCandUSART ;go StartADCandUSART
;SUBROUTINE
ADCLight
 BSF ADCON0,GO ;start A/D conversion
Wait
 BTFSC ADCON0,GO ;check if A/D conversion is done
 GOTO Wait
 MOVF ADRES,W ;move ADC data to W
 RETURN
Send
 BSF STATUS, RP0 ;select bank 1
 BTFSS TXSTA, 1 ;check if transmission is available
 GOTO Send
 BCF STATUS, RP0 ;select bank 0
 MOVWF TXREG ;move data to TXREG register
 RETURN
 END ;end line of the code
```

Appendix C. PIC Assembly Code for Bi-directional Serial Communication

```
;This code is used to run the light refraction test bed
;1. Generate pulse train to drive servomotor to a desired angle
;2. Turn on the Laser mounted on the arm of the servomotor
;3. Turn on the DC motor
;4. Measure the light intensity and linear position
;5. Convert sensor data into 8-bit A/D and send them to PC
;6. Stop DC motor if the limit switch at the end is pressed
;7. Relocate light sensor to the initial position
 LIST p=16f74
 INCLUDE "p16f74.inc"
 __CONFIG _CP_OFF & _WDT_OFF & _HS_OSC &
_PWRTE_ON ;configure PIC16F74
counter EQU 20h ;file address of counter var
iteration EQU 21h ;file address of iteration var
tempval EQU 22h ;file address of tempval var
```

```
iter EQU 23h ;file address of iter var
iter1 EQU 24h ;file address of iter1 var
iter2 EQU 25h ;file address of iter2 var
iter3 EQU 26h ;file address of iter3 var

 ORG 0 ;origin address is 0
 CLRF STATUS ;clear status register
 GOTO BootStart ;go to BootStart

BootStart
 BANKSEL PORTA ;select bank 0
 CLRF PORTA ;clear portA
 CLRF PORTB ;clear portB
 CLRF PORTC ;clear portC

 BANKSEL TRISA ;select bank 1
 MOVLW b'00000111'
 MOVWF TRISA ;set RA0, 1, and 2 as inputs
 MOVLW b'00000000'
 MOVWF TRISB ;set PORTB as all outputs
 MOVLW b'10000000'
 MOVWF TRISC ;set RC7 as input

ADCInitialization
 BCF STATUS, RP0 ;select bank 0
 MOVLW B'10000001'
 MOVWF ADCON0 ;enable ADC and select CH0
 BSF STATUS, RP0 ;select bank 1
 MOVLW b'00000100'
 MOVWF ADCON1 ;set RA0, 1, and 3 to A/D ports

TimerInitialization
 BSF STATUS, RP0 ;select bank 1
 MOVLW b'00000001'
 MOVWF OPTION_REG ;set prescaler of TMR0 to 1:4
 BCF STATUS, RP0 ;select bank 0
 MOVLW b'10000100'
 MOVWF INTCON ;enable all unmasked interrupts
                     ;and TMR0 register overflow

 CLRF TMR0 ;clear timer

BaudRateSettingsforUSART
 BSF STATUS, RP0 ;select bank 1
 MOVLW d'129'
 MOVWF SPBRG ;set baudrate 9600 for 20MHz crystal
 MOVLW b'00100100'
 MOVWF TXSTA ;8-bit asyn. high-speed transmission
 BANKSEL RCSTA ;select bank 0
 MOVLW b'10010000'
 MOVWF RCSTA ;8-bit asyn. continuous reception

 MOVF RCREG, W
 MOVF RCREG, W
 MOVF RCREG, W ;flush reception buffer 3 times

MainProgram
 CALL ReceiveAngle ;call ReceiveAngle subroutine
 MOVLW 0x64
```

```
 MOVWF iteration ;save iteration value for pulse train

BeginServo
 MOVF tempval, 0 ;move tempval to W
 MOVWF counter ;assign user input into counter

LoopHigh
 CLRF TMR0 ;clear timer
 BSF PORTB, 1 ;set RB1 to high
 MOVLW 0x05

InnerLoopHigh
 SUBWF TMR0, 0 ;set and countdown timer
 BTFSS STATUS, 2 ;check if timer is zero
 GOTO InnerLoopHigh ;go to InnerLoopHigh again
 BCF STATUS, 2 ;reset zero bit of status
 DECFSZ counter ;countdown counter and check if zero
 GOTO LoopHigh ;go to LoopHigh

 BCF PORTB, 1 ;set RB1 to low
 MOVLW 0xfa
 MOVWF counter ;set the value of counter for low

LoopLow
 CLRF TMR0 ;clear timer
 BCF STATUS, 2 ;reset zero bit of status
 MOVLW 0x15

InnerLoopLow
 SUBWF TMR0, 0 ;set and countdown timer
 BTFSS STATUS, 2 ;check if timer is zero
 GOTO InnerLoopLow ;go to InnerLoopLow again
 BCF STATUS, 2 ;reset zero bit of status
 DECFSZ counter ;countdown counter and check if zero
 GOTO LoopLow ;go to LoopLow

 BCF STATUS, 2 ;reset zero bit of status
 DECFSZ iteration ;countdown iteration, check if zero
 GOTO BeginServo ;go to BeginServo

TurnLaser
 BSF PORTB, 5 ;turn on the laser

TurnMotor
 BSF PORTB, 7 ;turn on the DC motor
 CALL DelayDCMotor ;call DelayDCMotor subroutine

StartADCandUSART
 CALL ADCLight ;call ADCLight subroutine
 CALL Send ;call Send subroutine
 CALL DelayUSART ;call DelayUSART subroutine
 CALL ADCPosition ;call ADCPosition subroutine
 CALL Send ;call Send subroutine
 CALL DelayUSART ;call DelayUSART subroutine
 BTFSS PORTA, 2 ;check if the light sensor is at the end
 GOTO StartADCandUSART ;go StartADCandUSART

ReverseDCMotor
 BCF PORTB, 5
 BCF PORTB, 7 ;stop the DC motor
 CALL DelayDCMotor2 ;call DelayDCMotor2 subroutine
```

```
 BSF PORTB, 6 ;reverse the direction of the DC motor
 CALL DelayDCMotor2 ;call DelayDCMotor2 subroutine
CheckInitialPosition
 BTFSS PORTA, 2 ;check if the sensor back to the origin
 GOTO CheckInitialPosition
Finish
 BCF PORTB, 6 ;turn off the laser
 GOTO Finish ;finish the program
;SUBROUTINE
ReceiveAngle
 BCF STATUS, RP0
 BCF STATUS, RP1 ;select bank 0
 BTFSS PIR1, RCIF ;check if data is received
 GOTO ReceiveAngle
 MOVF RCREG, W ;move received data to W
 MOVWF tempval ;save data from W into tempval
 RETURN
Send
 BSF STATUS, RP0 ;select bank 1
 BTFSS TXSTA, 1 ; check if transmission is available
 GOTO Send
 BCF STATUS, RP0 ;select bank 0
 MOVWF TXREG ; move data to TXREG register
 RETURN
ADCLight
 BCF STATUS, RP0 ;select bank 0
 MOVLW B'10000001'
 MOVWF ADCON0 ;enable ADC and select CH0
 CALL Pause ;call Pause subroutine
 BSF ADCON0, GO ;start A/D conversion
 GOTO Wait ;go to Wait
ADCPosition
 BCF STATUS,RP0 ;select bank 0
 MOVLW B'10001001' ;
 MOVWF ADCON0 ;enable ADC and select CH1
 CALL Pause ;call Pause subroutine
 BSF ADCON0, GO ;start A/D conversion
Wait
 BTFSC ADCON0, GO ;check if A/D conversion is done
 GOTO Wait
 MOVF ADRES, W ;move ADC data to W
 RETURN
Pause ;short delay
 MOVLW 08h
 MOVWF iter
Loop1 DECFSZ iter
 GOTO Loop1
 RETURN
DelayUSART ;delay for USART
 MOVLW 0x0a
 MOVWF iter1
 GOTO Delay
DelayDCMotor ;delay for limit switch in the beginnig
 MOVLW 0x1a
 MOVWF iter1
 GOTO Delay
DelayDCMotor2 ;delay for limit switch at the end
 MOVLW 0x3f
 MOVWF iter1
 GOTO Delay

Delay
Loop2 MOVLW 0xff
 MOVWF iter2
Loop3 MOVLW 0xff
 MOVWF iter3
Loop4 DECFSZ iter3
 GOTO Loop4
 DECFSZ iter2
 GOTO Loop3
 DECFSZ iter1
 GOTO Loop2
 RETURN
 END ;end line of the code
```

Acknowledgements

This work is supported in part by the National Science Foundation under grants 0227479 and 0337668 and the NASA/NY Space Grant Consortium under grant 39555-6519.

References

[1] Online: http://www.microchip.com/1010/index.htm, website of Microchip Technology, Inc.

[2] Online: http://www.microchip.com/1010/suppdoc/appnote/index.htm, website of Microchip Technology, Inc., (access link for application notes, code examples, and templates).

[3] D. W. Smith, *PIC in Practice*, Newnes, Oxford, U.K., 2003.

[4] Online: http://www.microchip.com/1010/pline/tools/picmicro/program/picstart/index.htm, website of Microchip Technology, Inc., (access link for PICSTART Plus Development Programmer).

[5] Online: http://www.olimex.com/dev/, website of Olimex Ltd., (access link for PIC-PG2B Development Programmer).

[6] Online: http://pdfserv.maxim-ic.com/en/ds/MAX220-MAX249.pdf, website of Maxim Integrated Products, (access link for MAX232 datasheet).

[7] Online: http://mechatronics.poly.edu/smart/, website of Polytechnic's NSF funded Research Experience for Teachers project.

[8] Online: http://www.microchip.com/download/lit/pline/picmicro/families/16f7x/30325b.pdf, website of Microchip Technology, Inc., (access link for PIC16F74 device datasheet).

[9] Online: http://www.mathworks.com/products/matlab/, website of The Math Works, Inc., developer and distributor of technical computing software Matlab (access link for Matlab product information).

[10] Online: http://www.mathworks.com/products/simulink/, website of The Math Works, Inc., developer and distributor of Simulink (access link for Simulink product information).

[11] Online: http://www.mathworks.com/products/dialsgauges/, website of The Math Works, Inc., developer and distributor of Dials and Gauges Blockset (access link for Dials and Gauges Blockset product information).

SANG-HOON LEE was born in Seoul, Korea. He received the B.S. degree in Mechanical Engineering from Sung Kyun Kwan University, Seoul, Korea, in 1996 and the M.S. degree in Mechanical Engineering from Polytechnic University, Brooklyn, NY, in 2002. From 1996 to 1997, he worked for Sam Sung Engineering Co., Ltd. in Korea. He is currently continuing research at Polytechnic University as a doctoral student. His research interests include linear/nonlinear control, UAV path planning and tracking control, and mechatronics.

YAN-FANG LI received the B.S. degree in materials science from Shanghai Jiao Tong University, China in 2000. She began pursuing the M.S. degree in the Department of Mechanical Engineering at Polytechnic University, Brooklyn, NY, in Spring 2002. Since Spring 2003, she has also worked as a teaching and research assistant with responsibilities in the area of mechatronics. She is expected to receive the M.S. degree in mechanical engineering in June 2004.

VIKRAM KAPILA is an Associate Professor of Mechanical Engineering at Polytechnic University, Brooklyn, NY, where he directs an NSF funded Web-Enabled Mechatronics and Process Control Remote Laboratory, an NSF funded Research Experience for Teachers Site in Mechatronics that has been featured on WABC-TV and NY1 News, and an NSF funded GK-12 Fellows project. He has held visiting positions with the Air Force Research Laboratories in Dayton, OH. His research interests are in cooperative control; distributed spacecraft formation control; linear/nonlinear control with applications to robust control, saturation control, and time-delay systems; closed-loop input shaping; spacecraft attitude control; mechatronics; and DSP/PC/microcontroller-based real-time control. He received Polytechnic's 2002 Jacob's Excellence in Education Award and 2003 Distinguished Teacher Award. He has mentored 38 high school students, 10 high school teachers, 7 undergraduate summer interns, and 5 undergraduate capstone-design teams and has supervised 2 M.S. projects, 2 M.S. thesis, and 2 Ph.D. dissertations.