Visual C++ Applications in an EET Curriculum

David R. Loker, P.E., Ronald P. Krahe, P.E.
Penn State Erie, The Behrend College

Abstract

In this paper, Visual C++ applications are presented that utilize the Microsoft Visual Studio .Net Integrated Development Environment (IDE). The Electrical Engineering Technology (EET) Baccalaureate program at Penn State Erie, The Behrend College, continues to expand to include computer technology elective courses. One of the courses proposed is an EET course in Windows software development. This is consistent with a goal within the EET program to introduce students to current software development tools, and to enable students to utilize software to control PC hardware devices such as data acquisition boards. Visual C++ is an environment that can be used to meet these requirements.

The prerequisite for the material presented within this paper for learning Visual C++ includes a solid foundation in C/C++. However, as C/C++ continues to be an industry standard programming language, there is a need to provide Visual C++ Windows programming exposure to students within EET curriculums to meet industrial requirements.

Within this paper, several proposed Visual C++ applications, designed to be included in a Visual C++ course, are presented. The applications are presented in increasing levels of difficulty to aid in learning the language. The first project is the creation of a virtual calculator. The objective of this project is to introduce Windows programming. A second project is used to introduce the programming of a PC data acquisition (DAQ) board. This experiment emphasizes acquiring analog input data and displaying the mean and AC RMS values of the resulting data.

There are several objectives for the presentation of these applications within this paper. It provides a resource to aid instructors in the development of a Visual C++ programming course within their EET curriculums. It presents several practical experiments that can be utilized to help shorten the learning curve required for this programming language. It introduces a Windows programming environment to students to help meet industrial requirements. Finally, it presents an example of the integration of PC hardware and software.

I. Introduction to Visual C++ Windows Programming

As an aid to the development of visual C++ programming, the Microsoft Foundation Class (MFC) Application Wizard is utilized.[1] The AppWizard asks a series of questions about the type of application and the features and functionality for the application. It then creates the shell of the application containing the user interface and the application code. Based upon the shell of the application, the programmer modifies the application by adding control objects to the user interface, attaching variables to the controls, adding event handlers with the appropriate code to the controls, and compiling and executing the application.

As an example to create a first project, an application called FirstWindow is created using the AppWizard. This application generates a user interface containing two command buttons, which are used to only terminate the application. The following procedure is used for creating this first project.

1.
2. After starting Microsoft Visual C++ .Net, click *Create New Project* and select *Visual C++ Projects* on the left side of the window and *MFC Application* on the right side of the window. Type the name of the project, which in this case will be "**FirstWindow**". Then, provide the location for the folder and select *OK*. This will start the AppWizard.
3. On the left side of the AppWizard window, select *Application Type* and select *Dialog based* under *Application type*. Leave the rest of the settings to their default values. A dialog-based application is much simpler than other applications but has no menus except the system menu and can't save or open a file.
4. On the left side of the AppWizard window, select *User Interface Features* and type "**This is my first window**" under the *Dialog title*.
5. Select *Finish* on the AppWizard window and the application is created. The environment is then returned to the Integrated Development Environment (IDE).
6. In order to compile the application, select *Build Solution* from the *Build* menu.
7. Select *Start* under the *Debug* menu to run the application. Figure 1 shows the dialog-based application executing. This application presents text with a TODO message and OK and Cancel buttons. The title bar contains the title entered from step "3" above. Either of the two buttons can be used to close the application. By clicking on the system menu on the top-left corner of the window, the About box is displayed. This is shown in Figure 2.

Figure 1. The FirstWindow Application

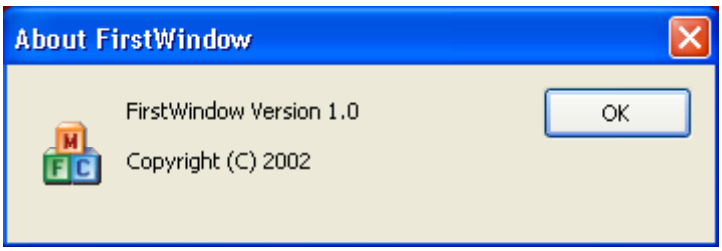

Figure 2. The About Box Window

Within the IDE, the Solution Explorer area on the right side of the window contains several panes for navigating the application code. The Class Pane contains the three classes that are created from the AppWizard. The CAboutDlg is a dialog class derived from the CDialog class and is used for the About dialog box. The CFirstWindowApp is a CWinApp class for the complete application. The CFirstWindowDlg is a CDialog class for the entire application. The Resource Pane allows for editing the About box window and the application window. The Solution Explorer Pane allows for editing all of the files that make up the application. These files consist of the Source files and the Header files. It also contains a ReadMe text file that describes all the files created by the AppWizard. The Solution Explorer Pane for this application is shown in Figure 3.

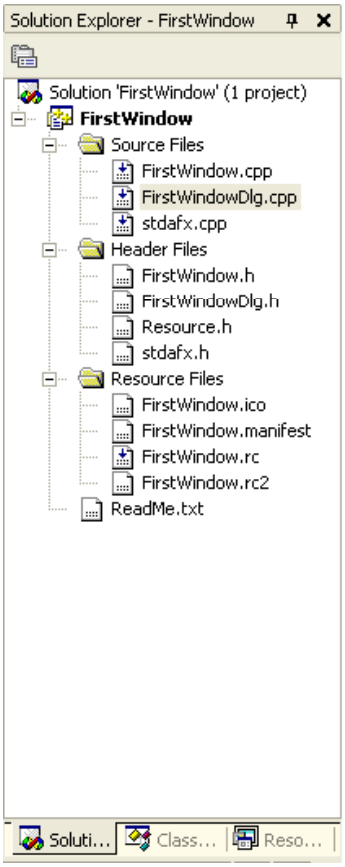

Figure 3. Solution Explorer Pane

II. Calculator Project

The first primary project discussed in this paper is the creation of a virtual calculator. The above-mentioned procedure for creating an application shell using the AppWizard is performed. The application shell is then modified by adding the windows controls, attaching variables to the controls so that a programmer can interact with them when writing the code, adding code to be used with the event handlers for the controls, and compiling and running the application.

II.A Windows Controls for the Calculator

The procedure for adding controls to the user interface is shown below. Before starting this procedure, select the TODO message displayed on the application window and delete it. Also, select the OK button and delete it as well. Detailed information on navigating the IDE can be found in one of the many references available.[2]

1. Add two edit box controls for the two inputs, one edit box control for the output, and one edit box for the operation performed on the two inputs. These controls allow the user to enter inputs for the application and to display outputs from the application. Change the property settings for each control as shown in Table 1. The ID property refers to the identification for the control so that code can be written to interact with the control. The number property restricts the text displayed within the control to numbers when the setting is TRUE.

Control	***Property***	***Setting***
Edit Box 1	ID Number	IDC_INPUT1 TRUE
Edit Box 2	ID Number	IDC_INPUT2 TRUE
Edit Box 3	ID Number	IDC_OUTPUT TRUE
Edit Box 4	ID	IDC_OPERATION

Table 1. Property Settings Changes for the Edit Box Controls.

2. Add one command button to allow the user to trigger an action. This command button will be used to initiate the operation performed on the two input numbers. Change the property settings as shown in Table 2. The caption property specifies the text displayed in the control.

Control	***Property***	***Setting***
Command button	ID Caption	IDC_EQUALS =

Table 2. Property Settings Changes for the Command Button.

3. Add five static text controls to label the edit box controls. These controls display text information to the user but do not allow the user to change or interact with them. Change the property settings as shown in Table 3.

Control	***Property***	***Setting***
Static Text 1	Caption	Input Number 1
Static Text 2	Caption	Input Number 2
Static Text 3	Caption	Output Result
Static Text 4	Caption	Operation (Ex: +,-,*,/)
Static Text 5	Caption	Depress the = button to execute

Table 3. Property Settings Changes for the Static Text Controls.

II.B Attaching Variables to the Controls

Before writing code for the application, it is necessary to assign variables to the controls on the user interface. This will be done to all of the controls except the static text controls and the command button. This is needed so that the programmer can interact with these variables as the code is written. The procedure for attaching a variable to each control is shown below.

1. Select the control for attaching a variable to, right-mouse click, and select *Add Variable*. This will bring up an add member variable wizard window to allow the programmer to add the appropriate information.
2. Enter the variable name, category, and variable type and then select *Finish*.

The information shown in Figure 4 is for the INPUT1 edit box control. The complete information for the variable associated with each control is shown in Table 4.

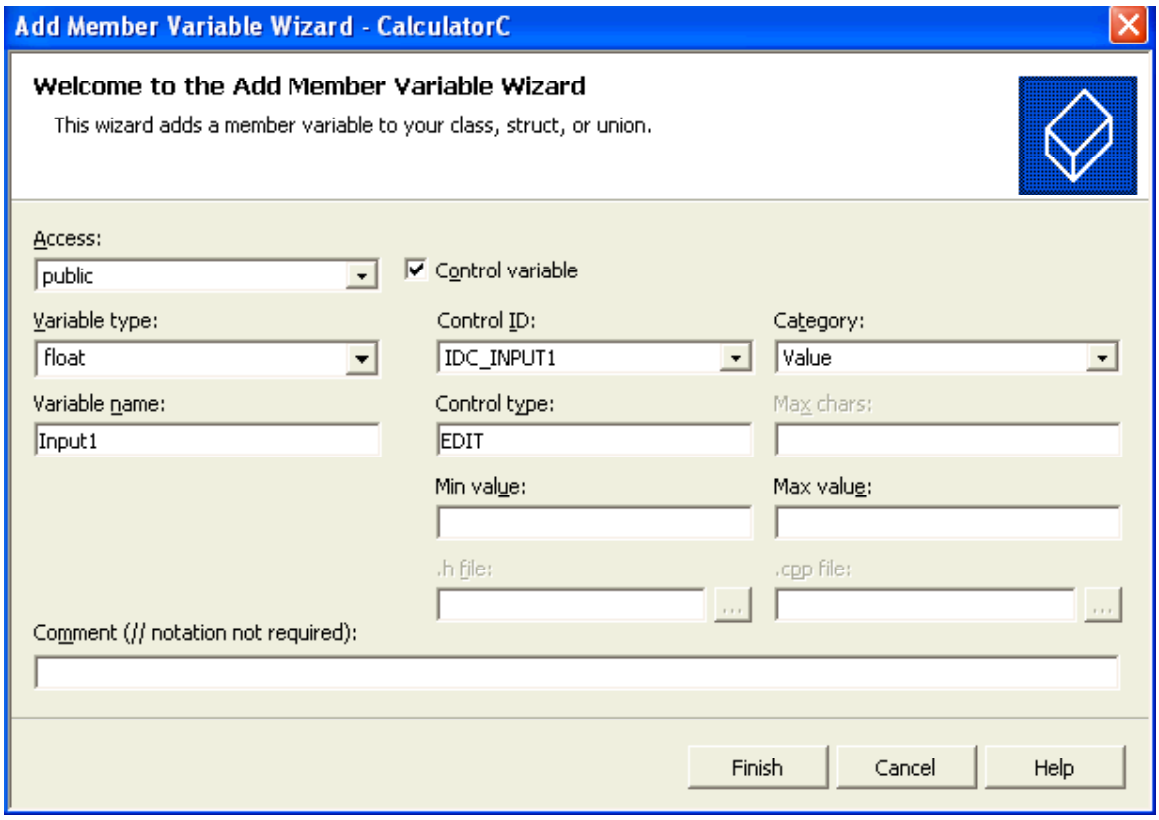

Figure 4. Add Member Variable Wizard for INPUT1 .

Control ID	***Variable Name***	***Category***	***Variable Type***
IDC_INPUT1	Input1	Value	float
IDC_INPUT2	Input2	Value	float
IDC_OUTPUT	Output	Value	float
IDC_OPERATION	Operation	Value	CString

Table 4. Attaching Variables to the Controls.

II.C Adding Event Handlers with the Appropriate Code

The EQUALS command button is depressed to perform the mathematical operation on the two input numbers and the result is displayed. In order to add functionality to the command button and add the proper application code, the following procedure is used.

1. Select the '=' button on the user interface.
2. In the Properties Pane, select the *Control Events* button (the lightening bolt).
3. Select *BN_CLICKED* in the message list. The value area beside the event should

become a combo box. Click the arrow and select *OnBnClickedEquals*.

4. The editor area will open within the event handler for the depressed '=' command button. This event handler is within the **CCalculatorCDlg** C++ program. Add the code shown below in Listing 1.

```
void CCalculatorCDlg::OnBnClickedEquals()
{
	// TODO: Add your control notification handler code here
	UpdateData(TRUE);

	switch(Operation[0]) {

		case '+': Output=Input1+Input2; break;

		case '-': Output=Input1-Input2; break;

		case '*': Output=Input1*Input2; break;

		case '/': Output=Input1/Input2; break;

	}

	UpdateData(FALSE);
}
```

Listing 1. C++ Code Listing for the Equals Event Handler.

The **UpdateData** function in Listing 1 takes the data in the variables and updates the controls on the user interface with the values of the variables, when the argument is FALSE. When the argument is TRUE, it takes the values from the controls and updates the variables with these values.

II.D. Adding Initialization Code

When the application is first executed, default values of 0 are displayed in the edit box controls used to display numerical values. For the operation edit box control, an initial value for this must be specified. This is provided by adding the following code within the BOOL **CCalculatorCDlg::OnInitDialog()** function as shown in Listing 2.

```
	// TODO: Add extra initialization here
	Operation="+";
	UpdateData(FALSE);
```

Listing 2. Initialization Code for the Edit Box Control.

Following this procedure, the application is compiled and run. The resulting user interface for this application is shown in Figure 5.

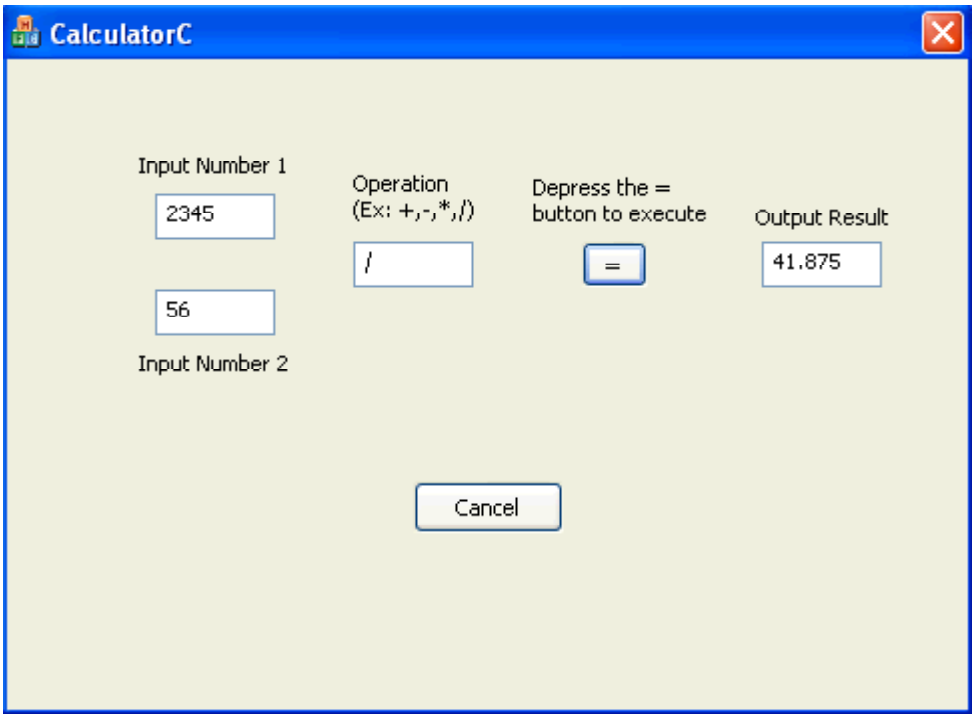

Figure 5. The Calculator Application.

III. Data Acquisition Project

The second project is the creation of a data acquisition (DAQ) project. The objective of this project is to create an application to acquire data from an analog input channel of the DAQ card, write the data to a text file, and display the mean and AC RMS values from the acquired data. The user inputs consist of the device number, the channel number, the number of scans to acquire, and the scan rate. The DAQ card utilized for this project is the Lab-PC-1200 DAQ card manufactured by National Instruments.[3] The procedure for creating this application is shown below.

1. Create an application shell using the AppWizard as discussed previously.
2. Add seven edit box controls to the user interface. The first four controls are for the device number, the channel number, the number of scans to acquire, and the scan rate. Two more controls are used to display the mean and AC RMS values. One additional control is used to display text comments to the user indicating the current state of the program. Change the property and settings for each control to reflect their function.
3. Add two command buttons to the user interface. One button is used to initiate the

acquisition of the data. A second button is used to terminate the application. Change the property and settings for each command button to reflect their function.

4. Add appropriate static text controls to the edit box controls to identify to the user their functions.
5. Attach the appropriate variables to each of the edit box controls.
6. Add the appropriate event handler to the acquire button. Then add the code to the application as shown in listing 3. The **AIAcquireWaveforms** function shown within the code takes the device number, the channel number, the number of scans, and the scan rate and returns the data in the variable name "**waveform**".[4]

```
_
void Cdaqv2Dlg::OnBnClickedButton1()
{
        // TODO: Add your control notification handler code here

        UpdateData(TRUE);
        static double waveform[10000];
        static double waveformNew[10000];
        static double sum;
        int count;
        FILE *fpt;

        static double actscanrate;
        static short error;
        char ChannelStr[2];

        sprintf(ChannelStr, "%s", Channel);

        error=AIAcquireWaveforms (Device, ChannelStr, NumScans, ScanRate, 0.0,
                0.0, &actscanrate, GROUP_BY_CHANNEL, waveform);

        // Write the data to the file 'wave.txt' and sum the data to calculate the mean
        fpt=fopen("wave.txt", "w");

        sum=0;
        for (count=0; count<NumScans; ++count)
        {
                fprintf (fpt, "%f\n", waveform[count]);

                sum+=waveform[count];
        }
        fclose(fpt);

        Mean=sum/NumScans;

        // Subtract the mean to calculate the AC RMS value.
        for (count=0; count<NumScans; ++count)
        {
                waveformNew[count]=waveform[count]-Mean;
        }
```

```
        sum=0;
        for (count=0; count<NumScans; ++count)
        {
              sum+=(waveformNew[count]*waveformNew[count]);
        }

        RootMeanSqr=sqrt(sum/NumScans);

        // Display the comment to the edit box control.
        CommentNew="Results stored in file named wave.txt.";

        UpdateData(FALSE);
}
```

Listing 3. C++ Code Listing for the Acquire Event Handler.

7. Special header and library files need to be added to the project in order to run the DAQ card. These are obtained from the National Instruments LabWindows/CVI software.[4] The header files that need to be included are the **easyio.h** and the **cvidef.h** files. The library files that need to be included are the **easyio.lib**, **cvisupp.lib**, and **cvirt.lib**. The easyio library and header files contain the functions for the DAQ card. The function used for this application is the **AIAcquireWaveforms** function shown within the code listing in Listing 3, which executes the DAQ card and returns an array of data. The other header and library files are associated with the LabWindows/CVI support functions. One additional header file included is the **math.h** file for the square-root function within Listing 3. The resulting Solution Explorer Pane for the application is shown in Figure 6.

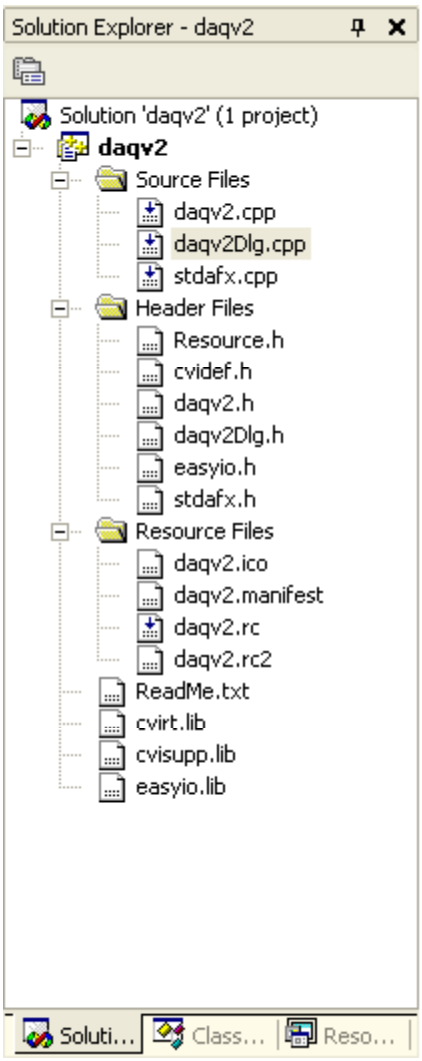

Figure 6. Solution Explorer Pane for the DAQ application.

8. Initialization code is used for the edit box controls under the **OnInitDialog()** function. This code is shown in Listing 4.

```
// TODO: Add extra initialization here
Device=1;
Channel="0";
NumScans=100;
ScanRate=1000;
CommentNew="Press the acquire button to capture data.";
UpdateData(FALSE);
```

Listing 4. Initialization Code for the Edit Box Controls.

The resulting user interface is shown in Figure 7. The waveform captured by the DAQ card was a 100Hz sinusoidal waveform with 2 volts p-p and 0 volts DC offset. The signal was applied to channel 0 of the DAQ card. The DAQ card acquired 100 samples at a sample rate of 1K samples/sec.

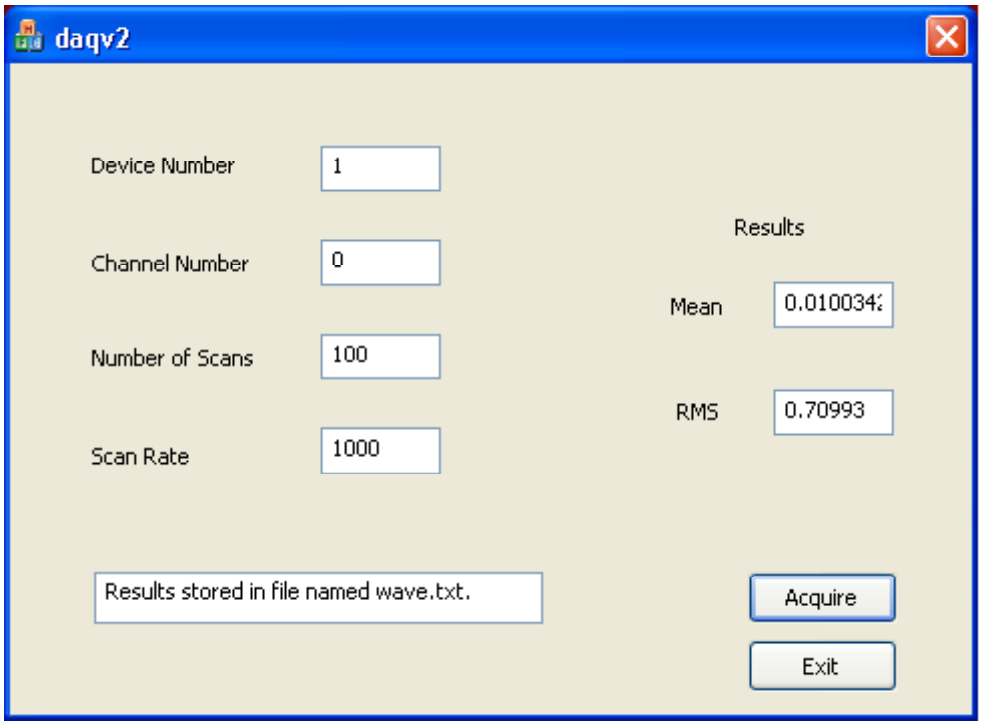

Figure 7. The DAQ Application.

IV. Conclusion

There are several objectives for presenting the Visual C++ projects within this paper. It provides a tutorial to help instructors in the development of several projects that can be utilized within a Visual C++ programming course. It presents several practical examples that can be used to help shorten the learning curve required for this programming language. It introduces a software development environment to students that is utilized to help meet the current needs within industry. Finally, it demonstrates an example of the integration of PC hardware and software.

Bibliography

1. Gregory, K., *Special Edition Using Visual C++ .NET*, Que, 2002.
2. Chapman, D., *Teach Yourself Visual C++ .NET in 21 Days*, Sams, 2002.
3. National Instruments, Lab-PC-1200 DAQ Card Manual, 1996.
4. National Instruments, LabWindows/CVI Programmer Reference Manual, 1998.

DAVID R. LOKER
David R. Loker received the M.S.E.E. degree from Syracuse University in 1986. In 1984, he joined General Electric (GE) Company, AESD, as a design engineer. In 1988, he joined the faculty at Penn State Erie, The Behrend College, in the Electrical Engineering Technology program. His research interests include PC-based control systems, communication systems, and instrumentation systems.

RONALD P. KRAHE
Ron Krahe has over 30 years industrial experience in product design and development related to embedded controls, sensors and instrumentation. He joined Penn State Erie in 1988, and currently holds the rank of Associate Professor of Engineering. His teaching specialties include computer programming & embedded controls, and electricity & electronics. His research interests include mechatronics, embedded controls, and sensors & signal processing.