Using principal component analysis for early fault detection in Louisville water distribution system.

Kevin M. Kastensmidt
University of Louisville

USING PRINCIPAL COMPONENT ANALYSIS FOR EARLY FAULT DETECTION IN LOUISVILLE WATER DISTRIBUTION SYSTEM

By

Kevin M. Kastensmidt
B.S., University of Louisville, 1995

A Thesis
Submitted to the Faculty of
University of Louisville
J. B. Speed School of Engineering
As Partial Fulfillment of the Requirements
For the Professional Degree

MASTER OF ENGINEERING

Department of Computer Engineering and Computer Science

December 2011

USING PRINCIPAL COMPONENT ANALYSIS FOR EARLY FAULT DETECTION IN LOUISVILLE WATER DISTRIBUTION SYSTEM

Submitted by:___________________________________

Kevin M. Kastensmidt

A Thesis Approved On

____

(Date)

by the Following Reading and Examination Committee:

____

Dr. Ahmed H. Desoky, Thesis Director

____

Dr. Adel S. Elmaghraby

____

Dr. John F. Naber

ACKNOWLEDGMENTS

My academic pursuits have been influenced by many people. At a young age, my parents showed me the value of a good education which developed into following my Father's footsteps to Speed School for an engineering degree. My wife, Amy, and children, Autumn, Trenton, and Carson, have been most supportive and patient while completing the Master's program. Their encouragement made this possible!

I would like to thank Louisville Water Company for supporting my efforts. In particular, Larry Bryant gave his time and expertise to explain the distribution system and SCADA environment as well as assist with interpretation of the data. He was able to shed light on many facets of my research.

I am very appreciative of Dr. Desoky. As my thesis director, he has been patient and helpful to guide the research direction. Due to work and family responsibilities, it has taken quite a while for me to complete the program. Throughout it all, Dr. Desoky has been very supportive.

ABSTRACT

This research dealt with the examination of Supervisory Control and Data Acquisition (SCADA) information of a water distribution system through Principal Component Analysis (PCA). PCA is a mathematical method to convert a set of possibly correlated data into a set of fewer variables called principal components. In a SCADA environment, possibly hundreds of data points such as booster pumps, storage tanks, pressure reducing valves, and others constantly provide operational statistics including water pressure, tank capacity, and more. This vast amount of data can be difficult to analyze in its entirety, especially to detect issues in the distribution system.

PCA was utilized to observe abnormalities in these SCADA readings. Each SCADA data point was used as an input variable to PCA such as the pressure flow through a pump. Various calculations could be achieved by examining data points from a specific pressure zone or through the entire system. Breaking down the observations into specific areas resulted in better identification of the problem location. Each SCADA data point also provides an updated reading each minute. At the same interval, the principal component is calculated along with the variance of the prior twenty minutes worth of data. The difference between the current variance and the previous minute's variance highlights possible issues when compared to normal operations. For instance, at 11 am, the current principal component is computed and the principal component results from 10:40 am to 11 am are used as inputs in determining the current variance. This variance, when compared to the previous minute's variance, is plotted to show deviations in the data. One principal component was calculated each minute resulting in a single value to correlate all provided inputs regardless of the number of SCADA data points analyzed.

Analysis of normal operations still results in varying outputs as a result of low and high demand on the water distribution system throughout the day but maintains a regular pattern. Review of data during a main break condition emphasizes the irregular pattern signaling a possible fault. PCA interpretation can be an additional monitoring tool of the distribution system to provide advanced warning of main breaks or other system issues.

TABLE OF CONTENTS

Page

APPROVAL PAGE …………………………………………………………...…………ii

ACKNOWLEDGEMENTS …………………………………………….…………..iii

ABSTRACT ……………………………………………………….………………iv

LIST OF TABLES …………………………………………………………….……vi

LIST OF FIGURES ……………………………………….……….………………vii

I. INTRODUCTION ……………………………………………..……..1

A. LOUISVILLE WATER COMPANY …………………………….2

B. PROBLEM STATEMENT ………………………………………….3

II. PRINCIPAL COMPONENT ANALYSIS …………….................……4

A. PCA EXAMPLE ……………………………………...…………..7

III. SCADA ENVIRONMENT ………………………...……………..13

IV. DATA ANALYSIS ………………………………………….………18

A. TEST ENVIRONMENT SETUP ……………………………….18

B. PRINCIPAL COMPONENT CALCULATION ……………...……25

C. PRINCIPAL COMPONENT OPERATION ……………………35

V. RESULTS …………………………………………………..….40

A. INITIAL TESTING ………………………………………………..40

B. ENTERPRISE ANALYSIS …………………………..……46

C. RUNTIME ENVIRONMENT ……………………………..….51

VI. CONCLUSIONS AND RECOMMENDATIONS ……………….…………57

APPENDIX I. MATLAB CODE TO INSERT MULTIPLE RECORDS ……………....60

APPENDIX II. MATLAB GUI CODE ………………………………….……..61

LIST OF REFERENCES …………………………………………………..……80

VITA ……………………………………………………….…………………81

LIST OF TABLES

Page

1. PCA EXAMPLE DATA ……………………………………………………….8
2. PCA EXAMPLE UPDATED RAW DATA ..…………………………………….8
3. PCA EXAMPLE NORMALIZED DATA ..……………………………………...9
4. PCA EXAMPLE FIRST PRINCIPAL COMPONENT ………………………...12
5. [SCADA].[dbo].[ihRawData] Table Structure ………………………………….19
6. [SCADA].[dbo].[PRESSURE_ZONES] Table Structure ………………….…20
7. PRESSURE_ZONES Table Sample …………………………………………….22
8. [SCADA].[dbo].[HistoricalPCA] Table Structure ……………………………....33
9. [SCADA].[dbo].[HistoricalPCA] Single Field Sample …………………………34
10. [SCADA].[dbo].[PZALL] Table Structure ……………………………………...35

LIST OF FIGURES

Page

1. Typical Historian System ……14
2. GIS Storage Tank Location Example ……16
3. SCADA System Overview ……17
4. SCADA Water Quality Information ……17
5. LWC Pressure Planes ……21
6. Update Historical PCA Multipliers GUI ……24
7. Message Box When Incorrect Time Data Entered ……25
8. Progress-Bar Illustration ……27
9. Brooks Hill 760 Pressure Zone Normal Operations ……40
10. Brooks Hill 760 Pressure Zone Break Condition ……41
11. PCA Calculation Example using 20 Minute Variance ……43
12. PCA Calculation Example with 60 Minute Variance ……44
13. PCA Calculation Example with 45 Minute Variance ……44
14. PCA Calculation Example with 15 Minute Variance ……45
15. Comparison of Variance Calculation ……46
16. Normal Operating Day ……47
17. First Large Break Occurrence ……48
18. Second Large Break Occurrence ……48
19. First Large Break Occurrence Detail ……49
20. Second Large Break Occurrence Detail ……50
21. Runtime Interface Starting View ……51
22. Runtime Pressure Zone Listing ……52
23. Sample Execution with Normal Operation ……55
24. Sample Execution with Break Condition ……55

I. INTRODUCTION

The primary objective of Principal Component Analysis (PCA) is to reduce the complexity of a set of data which includes a large number of related variables. The variation of the original data set should also be maintained as much as possible. This is accomplished by transforming the data to a new set of variables called principal components. These are unrelated and ordered such that the first few include most of the variation in all of the original variables [7].

The research involved very large datasets with a large number of variables from the Supervisory Control and Data Acquisition (SCADA) system at the Louisville Water Company (LWC). A SCADA infrastructure denotes an industrial control system that monitors and controls industrial, infrastructure, or facility-based processes [8]. The SCADA system controls and monitors the water treatment and distribution for Louisville Water Company. In other instances, SCADA can control or monitor wastewater collection ad treatment, electrical power transmission and distribution, or monitor building energy consumption [8].

The SCADA system is comprised of several components including a Human-Machine Interface (HMI), computer system, remote terminal units (RTU), programmable logic controllers (PLC), and a communication network. The HMI is the device that allows a human operator to interact with the system to monitor and control processing. The computer system retrieves and processes data as well as sends control signals throughout the system. RTU's connect to sensors to process and send signals to the computer system. PLC's are types of remote terminal units and are commonly used in the field because of their advantages as special-purpose remote terminal units. The communications network connects the field units, including the RTU's and PLC's with the computer system [8].

SCADA systems monitor and control sites spread out over a large area. This could include a single industrial plant or a larger geographical region such as a city or metropolitan area. In the field, control actions are performed by the RTU or PLC. For instance, a PLC can control and monitor the water level of a storage tank. The RTU or PLC takes meter or equipment readings which are transmitted over the communications network to the computer system. In the case of the storage tank, the water level is transmitted. Data from each of the systems RTUs or PLCs is compiled in the computer system. Typically, a database is used to store each of the data endpoints which are called tags. Each tag represents a single input or output from the RTUs or PLCs throughout the system. An operator is able to access this information through the HMI to gather data trends or make system changes such as opening or closing a valve. When the operator modifies the configuration through the computer system, the change is transmitted over the communications network and applied to the RTU or PLC [8].

LOUISVILLE WATER COMPANY A

Louisville Water Company (LWC) is a municipally owned corporation that provides water to about 850,000 people in a 600+ square mile area encompassing Louisville Metro/Jefferson County, KY and areas of Oldham and Bullitt Counties. LWC provides water for resale to Taylorsville, Mount Washington, North Shelby, and Salt River, KY. The distribution system consists of over 4,000 miles of water mains, over 23,000 hydrants, almost 50,000 system valves, and over 300,000 services [1,2].

Since its inception over 150 years ago, Louisville Water Company has been providing safe, high quality water. Very innovative processes have been created during this time including the redundancy and design of the distribution system, use of the water quality lab that now performs over 200 tests daily on the drinking water supply, and being the first water utility in the world to combine a tunnel with gravity-fed wells as a source for drinking water [9].

Part of what aids the innovation is the information provided by the SCADA system. It contains 9,000 data points, or tags, in the real-time database at 1 second intervals. Each tag is a unique identifier for a specific measurement attribute and corresponds to a field data point such as pressure through a valve or temperature of finished water [6]. The historical database stores over 850 of these tags at 1 minute intervals and has data back to 2003 [4].

PROBLEM STATEMENT B

There are vast amounts of data to analyze in LWC's complex environment. A highly qualified and dedicated staff maintains the SCADA infrastructure and distribution system. Proper maintenance and quick problem resolution depend on the staff understanding the system and being able to correlate data from several sources including SCADA, Geographical Information System (GIS), and historical trends. This requires a deep and extensive knowledge of the operating environment. A need exists to stay current on the status of daily operations including break locations and equipment maintenance schedules. The volume of data to process is also quite large. Looking at a single data point such as a tank level is simple to process but will only provide a limited amount of information. Looking at multiple data points such as all SCADA monitoring points in a pressure plane can provide numerous combinations of data and become labor intensive to study manually. Experienced employees are able to recognize certain trends in the raw SCADA readings that could be indicative of system issues. These readings are typically much closer to the climax of a break and do not provide much advanced warning.

The large number of SCADA variables and the constant influx of information make PCA a practical tool for examination. Using PCA with the SCADA information reduces the amount of data by computing a single principal component for multiple monitoring points thereby correlating the data automatically. Systems currently in place monitor the environment and provide alerts on various components in the distribution system including equipment failures like a large pumping station outage. PCA can extend alerting capabilities by providing earlier notification of issues in the pipe infrastructure. Observing subtle changes in the analysis can signal that a large main break or other event in the system could be pending.

II. PRINCIPAL COMPONENT ANALYSIS

The focus of PCA is the reduction of a complicated set of data without losing the variation of the original data. Applications for PCA can include pattern searching such as for facial recognition as well as image compression [10]. The initial details of the methods now known as PCA were provided by Pearson (1901) and Hotelling (1933). Research is still being completed in the general area of PCA. There is also widespread use of PCA in a number of different areas including biology, chemistry, genetics, quality control, and many more. This widespread use is evident by the 2000 articles published in the two years from 1999 to 2000 that contained the phrases 'principal component analysis' or 'principal components analysis' in the title, abstract, or keywords [7].

PCA can be calculated manually relatively easily if the number of variables, or dimensions, is small such as two or three dimensions. Once a data set has been chosen to have principal component analysis completed, the mean of the data points needs to be calculated. The mean, $\bar{x}$, is found by adding all numbers in the data set and dividing by the number of values in the data set, n, as shown by the formula

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \quad (1).$$

Numerous calculations throughout this process use the raw data minus the mean for each raw data point. It is beneficial to also compute this now [10].

Next, the standard deviation of the data set, represented by symbol σ, is computed. Standard deviation measures the spread of the data and is the average distance from the mean to a data point. It is derived by the formula

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{(n-1)}} \quad (2).$$

In the above formula, n represents the number of samples, x_i is the current raw data point, and $\bar{x}$ is the mean or average [10].

The normalization or standardization of the data is then calculated using the above information of mean and standard deviation. This normalization is the process of isolating statistical error in repeated measured data [11]. A SCADA system is an excellent example of a system with repeated measured data. The normalized data set is calculated by formula

$$\frac{x_i - \overline{x}}{\sigma} \tag{3}$$

for $1 \leq i \leq n$ where n is the number of items in the data set. Essentially, each raw data point has the mean subtracted and that value is divided by the standard deviation [10].

The next step in computing PCA is calculating the covariance matrix using the normalized data from above. Covariance measures the extent that variables vary with respect to each other. It is always measured between two variables. For data with three or more variables, covariance is calculated between each pair of variables. For example, data with three variables x, y, and z, covariance can be calculated between x and y, x and z, and y and z. For an m-dimensional set of data, the number of different covariance values that can be created is

$$\frac{m!}{(m-2)!*2} \qquad (4).$$

The formula for covariance between two normalized variables is

$$\operatorname{cov}(x, y) = \frac{\sum_{i=1}^{n} x_i y_i}{n-1} \tag{5}$$

where n is the number of data in the set. The covariance matrix is then a matrix of all possible covariance values between all of the different dimensions. The definition of the covariance matrix C with m variables is

$$C_{ij} = \mathrm{cov}(\mathrm{Var}_i, \mathrm{Var}_j) \qquad (6)$$

where $1 \leq i,j \leq m$ and Varx is the xth variable. From the definition, the covariance matrix will always be a square matrix. Continuing the example of data with three dimensions x, y, and z, the covariance matrix C would consist of three rows and three columns such as

$$C = \begin{bmatrix} \mathrm{cov}(x,x) & \mathrm{cov}(x,y) & \mathrm{cov}(x,z) \\ \mathrm{cov}(y,x) & \mathrm{cov}(y,y) & \mathrm{cov}(y,z) \\ \mathrm{cov}(z,x) & \mathrm{cov}(z,y) & \mathrm{cov}(z,z) \end{bmatrix} \qquad (7).$$

It can be noted that the main diagonal consists of the covariance with one of the variables and itself. These are actually the variances for that variable [10].

Once the covariance matrix is determined, its eigenvectors and eigenvalues can be discovered. An eigenvector is a vector that, when acted upon by a particular linear transformation, produces a scalar multiple of the original vector. The scalar in question is the eigenvalue that corresponds to the eigenvector [12]. Eigenvectors and eigenvalues can be expressed in terms of A which is a square matrix and a non-zero eigenvector x of A if an eigenvalue λ exists such that

$$Ax = \lambda x \qquad (8)\ [13].$$

When calculating the PCA, the square matrix in the above expression, A, is the covariance matrix C. It should be noted that eigenvectors can only be found for square matrices although not every square matrix has eigenvectors. Given any n x n matrix that does have eigenvectors, there are n eigenvectors. All eigenvectors of a matrix are also perpendicular to each no matter how many variables are involved. This allows the data to be expressed in terms of these eigenvectors. When eigenvectors are discovered, it is also desired to find ones that have a length of exactly one. The length of a vector doesn't affect whether or not it is an eigenvector but the direction of the vector does. To be standard, when eigenvectors are found, they are scaled to have a length of one. By the expression above, it is given that when an eigenvector is discovered, so is the eigenvalue [10].

The eigenvector with the highest eigenvalue turns out to be the primary component of the data set. This is easily seen once the eigenvectors found in the covariance matrix are ordered by eigenvalue from highest to lowest putting them in order of significance. At this time, components of lesser significance can also be ignored. This does cause a loss of information but if the eigenvalues are small it can be minimal. Also, if some information is removed, there will be fewer dimensions than the original data. A feature vector or a matrix of vectors is then constructed by putting the selected eigenvectors into matrix format with the eigenvectors in columns [10].

Finding the principal components is the last step. This is accomplished by dot-multiplying each column of the feature vector of eigenvectors by each row of the normalized data. This will provide an updated matrix of data that contains the principal components. The first column of data is the first principal component; the second column is the second principal component and so on. This gives the original data in terms of the vectors we have chosen. Since eigenvectors are perpendicular to each other as stated previously, the data can be represented by the axes of the final data set. This creates a transformed data set that is expressed in terms of the patterns between the data where the patterns are lines that describe the relationship between the data. The data values also shows how the data points compare to the trend, whether or not they are above or below the trend line seen from the data [10].

PCA EXAMPLE A

The following example illustrates the above steps to calculate principal components. As stated previously, PCA can be calculated easier with a smaller number of variables. For this reason, data was chosen with two variables. Also, the use of computer-based software allows for increased complexity and improved performance. This research was accomplished using Matlab software.

Data for this example consists of the following values which were generated by the Matlab random number function randn(10,1).

TABLE I

PCA EXAMPLE DATA

x	y
-0.0301	1.5326
-0.1649	-0.7697
0.6277	0.3714
1.0933	-0.2256
1.1093	1.1174
-0.8637	-1.0891
0.0774	0.0326
-1.2141	0.5525
-1.1135	1.1006
-0.0068	1.5442

The sum of the x values is -0.4854 and sum of the y values is 4.1669. Taking the sum of the raw data and dividing by the number of data points, 10, provides the mean values. These are -0.04854 for $\bar{x}$ and 0.41669 for $\bar{y}$.

The updated raw data with the mean subtracted is

TABLE II

PCA EXAMPLE UPDATED RAW DATA

$x - \bar{x}$	$y - \bar{y}$
0.01844	1.11591
-0.11636	-1.18639
0.67624	-0.04529
1.14184	-0.64229
1.15784	0.70071
-0.81516	-1.50579

0.12594	-0.38409
-1.16556	0.13581
-1.06496	0.68391
0.04174	1.12751

Using the expression provided above for standard deviation yields 0.8360 for the x variable and 0.9268 for the y variable.

The normalization of the data which consists of each raw data point subtracting the mean and that result divided by the standard deviation has the following results.

TABLE III

PCA EXAMPLE NORMALIZED DATA

x_N	y_N
0.022057	1.204037
-0.13918	-1.28008
0.808882	-0.04887
1.365808	-0.69301
1.384946	0.756047
-0.97505	-1.62471
0.150643	-0.41442
-1.39418	0.146535
-1.27385	0.737921
0.049927	1.216553

The normalized data above is used when calculating the covariance matrix. Since there are 2 variables, it will be a 2 x 2 matrix. By using the expression above for this matrix, it can be expressed as

$$C = \begin{bmatrix} \operatorname{cov}(x_{N,} x_N) & \operatorname{cov}(x_{N,} y_N) \\ \operatorname{cov}(y_{N,} x_N) & \operatorname{cov}(y_{N,} y_N) \end{bmatrix} \tag{9}$$

resulting in matrix

$$C = \begin{bmatrix} 1.000 & 0.0782 \\ 0.0782 & 1.000 \end{bmatrix} \qquad (10).$$

The covariance matrix C is then applied to equation 8 to discover eigenvectors and eigenvalues. In this equation, A is replaced by the covariance matrix C yielding

$$Cx = \lambda x \qquad (11).$$

As C is a 2 x 2 matrix, and since eigenvalues did exist, there are two. The two resulting eigenvectors from this covariance matrix are calculated by using equation 11 to be

$$\begin{bmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \end{bmatrix} \qquad (12).$$

The two resulting eigenvalues are

$$\begin{bmatrix} 1.0782 \\ 0.9218 \end{bmatrix} \qquad (13).$$

Breaking this down, the first eigenvector is

$$\begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} \qquad (14)$$

with an eigenvalue of 1.0782. The second eigenvector is

$$\begin{bmatrix} -0.7071 \\ 0.7071 \end{bmatrix} \qquad (15)$$

with an eigenvalue of 0.9218. These can be also be verified by using equation 11. As an example, verifying the first eigenvector and eigenvalue result in

$$\begin{bmatrix} 1.000 & 0.0782 \\ 0.0782 & 1.000 \end{bmatrix} \text{x} \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} = 1.0782\text{x} \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} \qquad (16).$$

This reduces to the following which validates the eigenvector and eigenvalue.

$$\begin{bmatrix} 0.7624 \\ 0.7624 \end{bmatrix} = \begin{bmatrix} 0.7624 \\ 0.7624 \end{bmatrix} \qquad (17).$$

The second eigenvector and eigenvalue can be verified using the same method.

The eigenvector with the highest eigenvalue is therefore the first eigenvector

$$\begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} \qquad (18)$$

since it had the eigenvalue of 1.0782 which is greater than 0.9218. The eigenvector with the highest eigenvalue is used to determine the first principal component. The eigenvector with the second highest eigenvalue is used to determine the second principal component, and so on with the remaining eigenvectors.

The principal components are then calculated from the normalized data and eigenvectors. Each row of the normalized data is multiplied by the chosen eigenvector column. This results in a single computed variable representing the principal component. For our example, the first row of data in the normalized data which includes values 0.0221 and 1.2040 is multiplied by the first eigenvector which includes 0.7071 and 0.7071. Using matrix multiplication, the first value becomes

$$[0.0221 \quad 1.2040] \times \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} = 0.8670 \qquad (19).$$

Completing the calculations on the remaining normalized rows produces the following results.

TABLE IV

PCA EXAMPLE FIRST PRINCIPAL COMPONENT

1st PCA
0.8670
-1.0036
0.5374
0.4757
1.5139
-1.8383
-0.1865
-0.8822
-0.3790
0.8955

This process can be repeated for the other eigenvector to calculate the second principal component if desired. As stated previously, these principal components can be plotted to show patterns that describe the relationship between the data and highlight their similarities and differences. It shows whether or not the data is consistent with a defined trend. As eigenvectors are orthogonal to each other, the data is shown in terms of the eigenvectors instead of the normal axes [10].

It is easily illustrated from the example that computing principal components is straight-forward, especially with a small number of variables. With a larger set of variables, these calculations become much more complex and more difficult to calculate manually. Computer software such as Matlab greatly simplifies this process.

III. SCADA ENVIRONMENT

In the late 1970's and early 1980's, LWC did not have a centralized SCADA system. A local Bristol-Babcock Digital Control System (DCS) was in operation at the B.E. Payne Facility. The Crescent Hill Filter Plant, Zorn Avenue Pumping Station, and Crescent Hill Pumping Station utilized local single-loop controllers for flow control and electro-mechanical drum-style controllers for automated tasks that involved timing or sequencing such as filter backwashing [4].

In the 1980's and 1990's, LWC upgraded to a Texas Instrument (TI) DCS system which was the company's first modern SCADA system that combined all of the major production facilities. Several generations of TI controllers including PM550 and TI500 series and HMIs handled controls, monitoring, and small amounts of data collection [4].

Beginning in early 2002, LWC had a complete upgrade of its automation and centralized control scheme. The local controllers were converted to Allen-Bradley's RSLogix platform. Each treatment facility was converted to Ethernet for local communications and Ethernet via Frame-Relay for remote distribution systems for communications [4].

The top-level SCADA was converted to Intellution iFix which is now a General Electric Intelligent Platforms product. The system consists of the following major components:

- Long-term archives running General Electric's iHistorian
- Microsoft Windows Servers consisting of redundant servers at different locations and all running General Electric's Proficy Server
- Microsoft Windows Workstations running General Electric's Proficy HMI
- RSLogix 5000 Series PLCs
- SLC PLCs
- MicroLogix PLCs [4]

The system today is a powerful, enterprise-wide data historian that retrieves, archives and distributes tremendous volumes of real-time production information at extremely high speeds. The architecture consists of several components including the Historian server, collectors, and clients. The Historian server is the central point for managing all of the client and collector interfaces, storing and (optionally) compressing data and retrieving data. All tag data is stored in a proprietary format in the archive database. The archive database consists of several files, each of which represents a specific time period of historical data. The system includes several types of data collectors for collecting data from a wide variety of applications. All client applications retrieve archived data through the Historian API. The Historian API is a client/server programming interface that maintains connectivity to the Historian Server and provides functions for data storage and retrieval in a distributed network environment. A typical Historian environment has the layout described in the following figure.

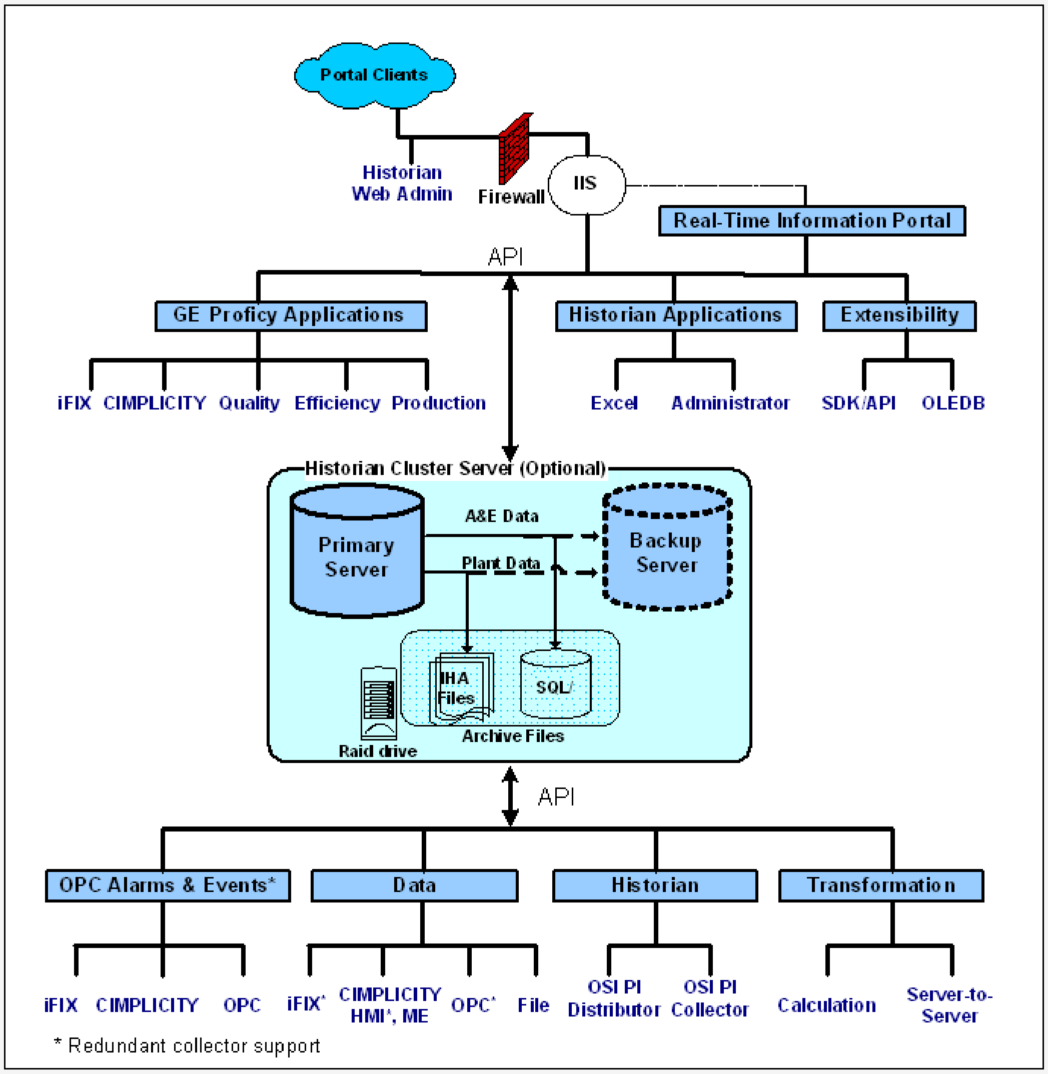


FIGURE 1 - Typical Historian System [6]

The Historian database tables contain read-only information from the Historian archive. Some of the tables included in the database are:

- ihTags – Contains tag configuration information.
- ihArchives – Contains archive filename and configuration information.
- ihCollectors – Contains configuration and status information for each collector.
- ihMessages – Contains messages such as alerts held in the audit log.
- ihRawData – Contains collected data for each tag.

The primary tables utilized in this study are the ihTags and ihRawData tables. Although the ihTags table contains numerous columns including the collector name and type used, input high and low scales, and many others, this research only utilized the tag name information. These tags represent various components of the distribution system including elevated storage tanks or pressure reducing valves to name a few. Many of these tags are represented in our GIS. Although there is currently no direct connectivity between GIS and SCADA systems for data value sharing, the GIS system does provide location information. The following illustration shows the area around eastern Jefferson County and Oldham County. It was created from the GIS data using ESRI ArcMap software. To simplify the map, only storage facility information is displayed which includes various ground and elevated storage tanks.

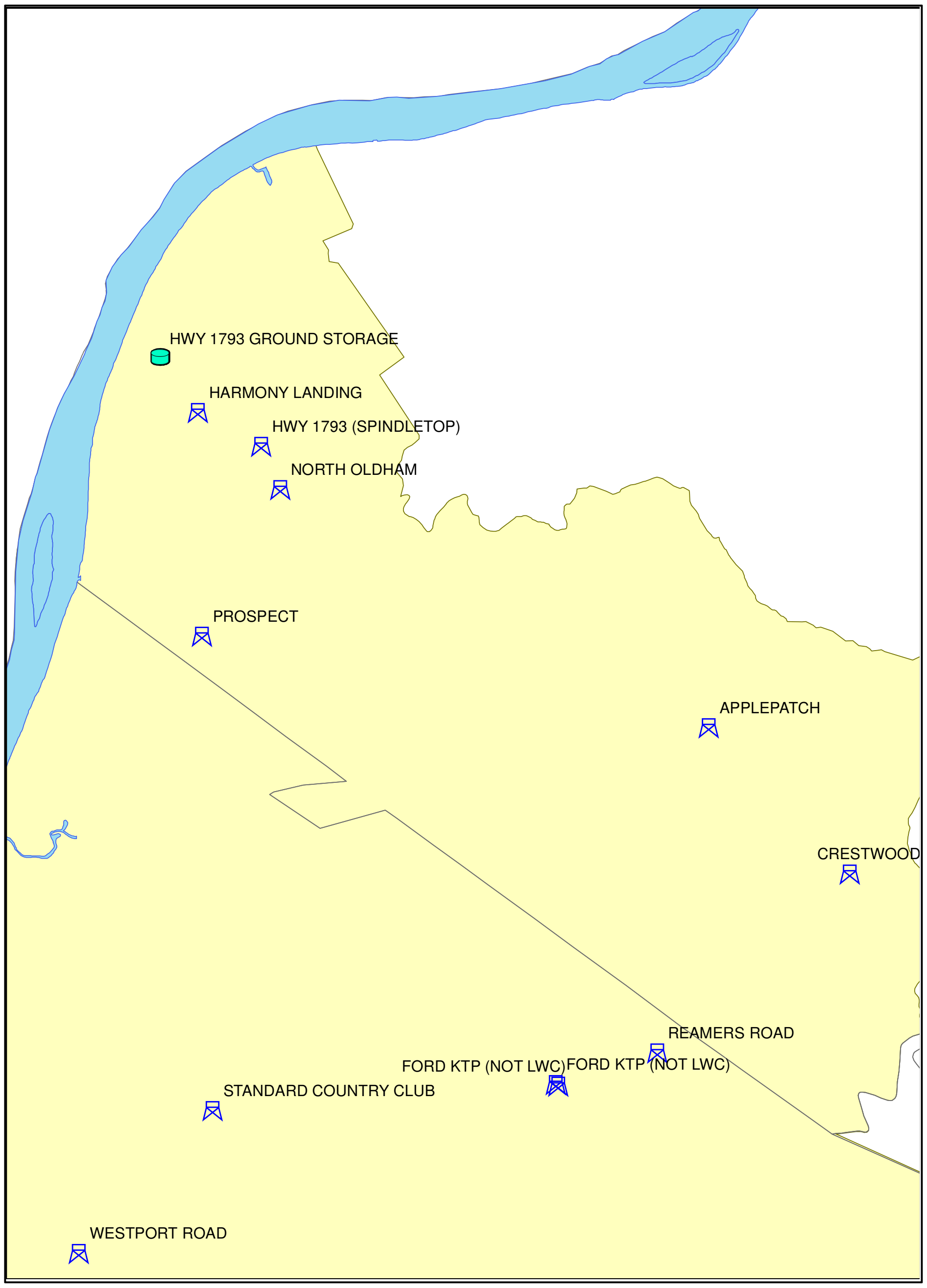
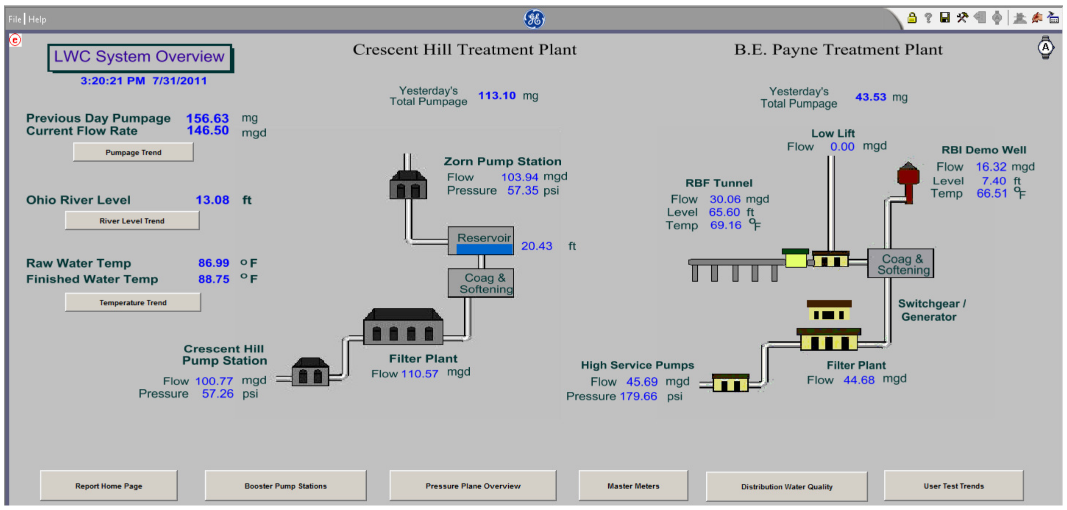


FIGURE 2 - GIS Storage Tank Location Example

Employees access SCADA data in several different ways depending on need. The SCADA administrator has created a web site based on the Proficy Real Time Information Portal. This provides quick and easy access for an employee to view current statistics relating to the distribution system including a general system overview as well as water quality data as an example.



LWC System Overview
3:20:21 PM 7/31/2011

Previous Day Pumpage 156.63 mg
Current Flow Rate 146.50 mgd

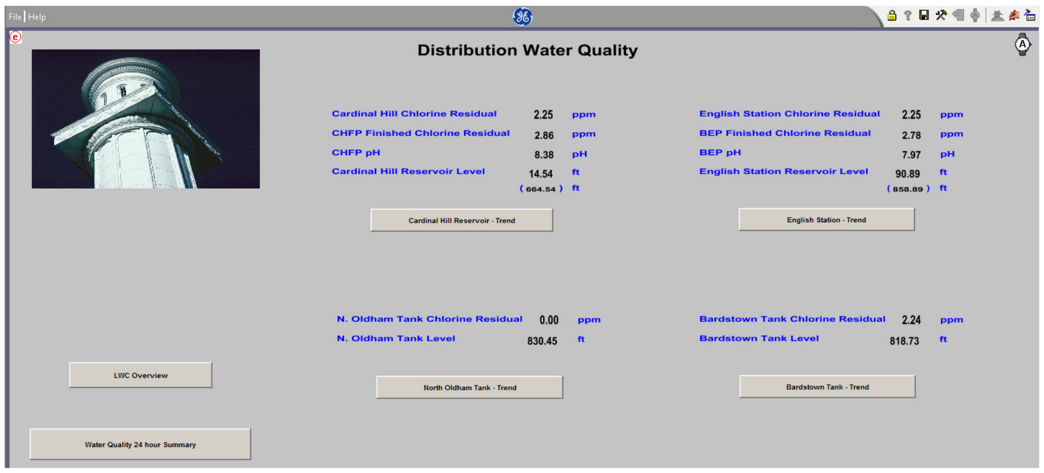
Ohio River Level 13.08 ft

Raw Water Temp 86.99 °F
Finished Water Temp 88.75 °F

Crescent Hill Treatment Plant
Yesterday's Total Pumpage 113.10 mg
Zorn Pump Station Flow 103.94 mgd Pressure 57.35 psi
Reservoir 20.43 ft
Coag & Softening
Filter Plant Flow 110.57 mgd
Crescent Hill Pump Station Flow 100.77 mgd Pressure 57.26 psi

B.E. Payne Treatment Plant
Yesterday's Total Pumpage 43.53 mg
Low Lift Flow 0.00 mgd
RBI Demo Well Flow 16.32 mgd Level 7.40 ft Temp 66.51 °F
RBF Tunnel Flow 30.06 mgd Level 65.60 ft Temp 69.16 °F
Coag & Softening
Switchgear / Generator
High Service Pumps Flow 45.69 mgd Pressure 179.66 psi
Filter Plant Flow 44.68 mgd

FIGURE 3 - SCADA System Overview


File | Help
Cardinal Hill Reservoir - Trend
English Station - Trend
LWC Overview
North Oldham Tank - Trend
Bardstown Tank - Trend
Water Quality 24 hour Summary

Distribution Water Quality

Cardinal Hill Chlorine Residual	2.25	ppm
CHFP Finished Chlorine Residual	2.86	ppm
CHFP pH	8.38	pH
Cardinal Hill Reservoir Level	14.54	ft
	(664.54)	ft
English Station Chlorine Residual	2.25	ppm
BEP Finished Chlorine Residual	2.78	ppm
BEP pH	7.97	pH
English Station Reservoir Level	90.89	ft
	(858.89)	ft
N. Oldham Tank Chlorine Residual	0.00	ppm
N. Oldham Tank Level	830.45	ft
Bardstown Tank Chlorine Residual	2.24	ppm
Bardstown Tank Level	818.73	ft

FIGURE 4 - SCADA Water Quality Information

Employees can also directly access the SCADA information through a Microsoft Excel Add-In which is a part of the Historian product. The Add-In allows access to any of the archive files for any of the tag and raw data information. The user can then create reports or perform needed analysis using standard corporate software [5].

IV. DATA ANALYSIS

TEST ENVIRONMENT SETUP A

To not affect the production SCADA system, data was copied to a local PC for data processing and analysis. The local PC is a Dell Latitude E6500 with 3 GB memory and a 2.67 GHz dual-core Intel processor running Microsoft Windows 7 SP1 operating system. Matlab R2010a, version 7.10.0.499, was installed for mathematical processing as well as creation of a graphical user interface (GUI). Microsoft SQL Server Express version 2008 R2 was installed as the database to store copied SCADA data as well as other tabular information required for PCA. The instance is named SQLEXPRESS.

Connectivity to the SCADA system needed to be established through the corporate network. The corporate network and SCADA networks are separated by a firewall for security purposes. Rules are in place such that corporate systems including desktops and servers can access SCADA servers. Servers and devices in the SCADA network require rules in the firewall to initiate connectivity. By default, none of the SCADA systems including servers and workstations can initiate network traffic through the firewall to the corporate network.

The Historian OLE DB Provider was next installed on the local PC so the SQL Server could have direct access to the Historian archives. The Historian OLE DB Provider is a data access mechanism that allows Historian data to be directly queried using Structured Query Language (SQL) statements [5]. This software is included with the regular Historian client. It allows the Historian server to be configured as a linked server in the SQL Server database. A linked server is a remote server defined in the SQL Server database through an OLE DB Provider. This allows tables and views to be defined in the local SQL Server that appear local to end users but actually access the data in the remote server. Although the installation and configuration of the Historian OLE DB Provider is very straightforward, the Historian server would not initially work as a linked server. When defining the linked server, the error message “Cannot create an instance of OleDb provider IhOLEDB.iHistorian for linked server ‘IHIST’”. This error was corrected after searching the GE Fanuc support site for a solution that consisted of adding registry key HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Microsoft SQL Server/MSSQL.1/Providers/iholedb.iHistorian. In this key, two DWORD values AllowInProcess and LevelZeroOnly were added and both set to one.

Once connectivity could be established, the raw SCADA data could be copied to the local SQL server. A SCADA database was also created in the local SQL Server instance. A local version of the ihRawData table was created in the local SCADA database as table [SCADA].[dbo].[ihRawData]. The local ihRawData table has the following table structure to match the Historian structure.

TABLE V

[SCADA].[dbo].[ihRawData] Table Structure

COLUMN_NAME	DATA_TYPE	CHARACTER_MAXIMUM
TagName	Nvarchar	4000
TimeStamp	Datetime	NULL
Value	Nvarchar	4000
Quality	Nvarchar	4000

Data from the Historian ihRawData archive is imported into the local table by using a SQL command in the SQLEXPRESS instance similar to the following.

```
insert into [SCADA].[dbo].[ihRawData] (TagName, TimeStamp, Value, Quality)
SELECT TagName, TimeStamp, Value, Quality FROM OPENQUERY
(iHist,'
SET
StartTime="09/08/2010 23:59:00",
EndTime="09/11/2010 23:59:59",
IntervalMilliseconds=1Minute,
SamplingMode=Calculated,
CalculationMode=Maximum
SELECT * FROM ihRawData WHERE TagName LIKE
CHFP.BROOKS_STATION_TANK_LEVEL.F_CV')
```

The above query contains the linked server, iHist, as the target of the OLE DB OPENQUERY function. The StartTime and EndTime parameters can be modified to import data from the desired timeframe. The IntervalMilliseconds parameter set to 1Minute tells the query to get a sample for each minute during the start and stop times provided. In the SELECT statement of the OPENQUERY function, the tagname is specified to select data for the appropriate tag. This query can be modified as needed to get the raw data from the Historian SCADA system.

When analyzing LWC data, it was initially decided to look at pressure plane boundaries to segment data. These geographic regions each run at the same elevation or plane. Each pressure plane contains components such as valves and storage tanks that have representative SCADA tags associated. Results and information regarding the evaluation of data is discussed further in the report.

As stated, each pressure plane would therefore contain components such as storage tanks or booster pumps to maintain the proper pressure. Another local table in the SQL Server Express SCADA database was created to link the pressure plan information with the SCADA tags in each zone. This table is named PRESSURE_ZONES. It contains the pressure zone name as required for this research, SCADA tag, pressure plane name, and facility identifier. The facility identifier is used as link for the GIS system and would provide a future interface for this research. Since multiple tags are in each pressure plane, this table has a one-to-many relationship between the pressure plane and the SCADA tags.

TABLE VI

[SCADA].[dbo].[PRESSURE_ZONES] Table Structure

COLUMN_NAME	DATA_TYPE	CHARACTER_MAXIMUM
PressureZone	Nvarchar	200
Tagname	Nvarchar	4000
PressurePlaneName	Nvarchar	200

FACILITY_ID	Nvarchar	50

These correspond to the tagname field in the PRESSURE_ZONES table. The PressureZone field in the table contains the name of the pressure zone in the LWC distribution system. Geographically, the pressure zones have the following layout.

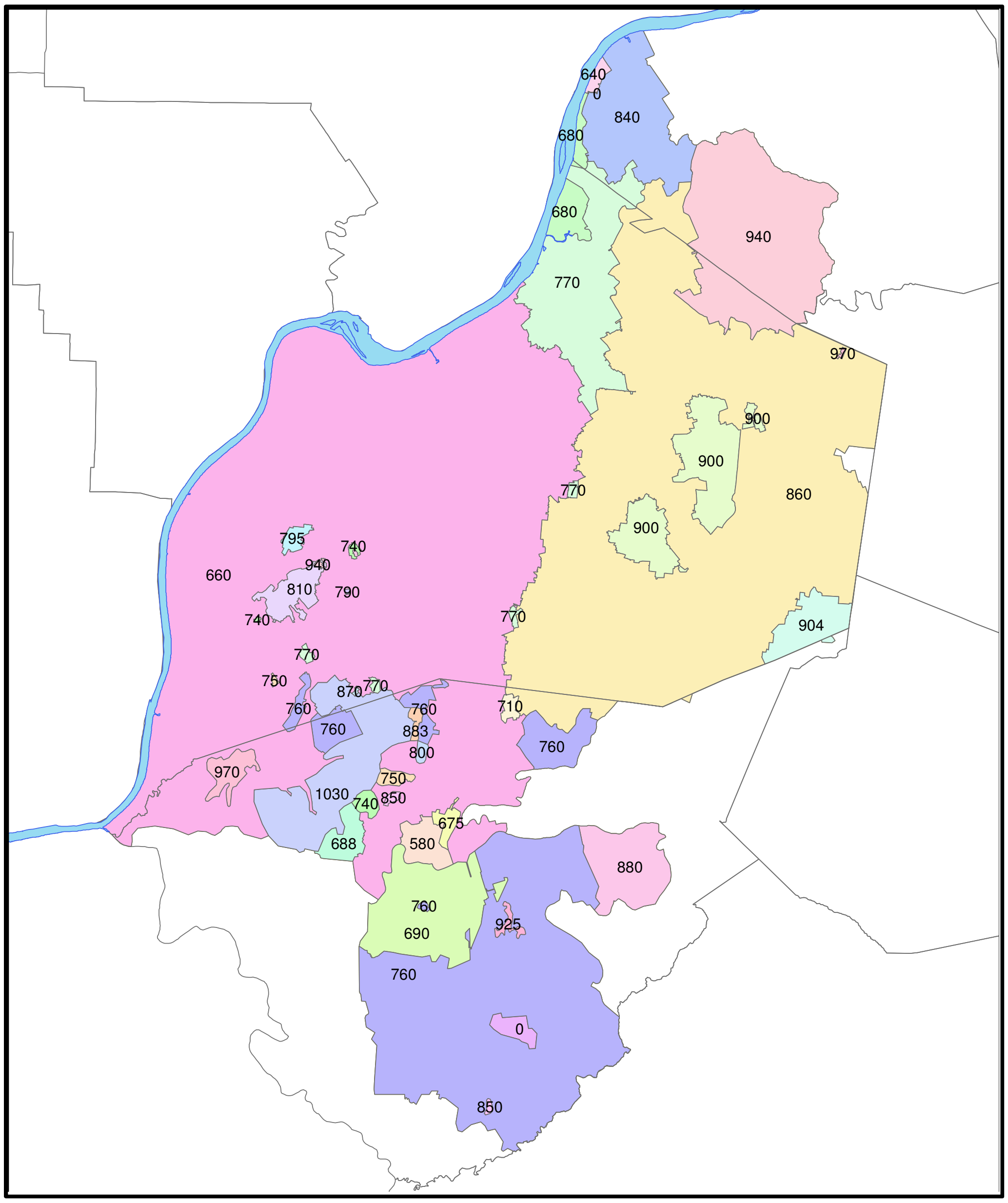


FIGURE 5 - LWC Pressure Planes

As an example, one of the 760 pressure zones contains four SCADA components or tags. The entries in the PRESSURE_ZONES table for these items include the following for the PressureZone and Tagname fields.

TABLE VII

PRESSURE_ZONES Table Sample

PressureZone	Tagname
760	CHFP.BROOKS_STATION_TANK_LEVEL.F_CV
760	CHFP.JEFF_DISCHARGE_PRESSURE.F_CV
760	CHFP.JEFF_FOREST_TANK_LEVEL.F_CV
760	CHFP.MARTIN_STATION_TANK_LEVEL.F_CV

As described previously, the raw SCADA data is loaded from the iHistorian system to the local SQL Server Express database. To expedite and automate this data load, the Matlab software package is utilized. At the start of a new Matlab program, BeginDate and EndDate variables are established to set the dates and times of data that will be transferred. These variables are in YYYY-MM-DD HH:MM format such as ‘2011-03-10 00:00’ for easier interaction with the SQL Server database. These dates can be days apart or any amount of time. However, the code loops in 24 hours intervals meaning that if the end date is only 1 hour greater than the start, there will still be 24 hours of data collected. The code connects to the SQLEXPRESS instance and queries the PRESSURE_ZONES table for all tags in a particular pressure zone. This creates a list variable containing the tags that will have data copied. For instance, it can contain ‘TagName = CHFP.JEFF_DISCHARGE_FLOW.F_CV or TagName = CHFP.BROOKS_DISCHARGE_PRESSURE.F_CV’. The code also takes into account whether or not it was the first tag in the list so it would not include the ‘or’ statement. An outer loop is created such that the query increments in 24 hour intervals between the start and stop dates. For example, if the start time is ‘2011-03-10 0:00’ and the stop time is '2011-03-30 23:59', the first iteration will only include the time between ‘2011-03-10 0:00’ and ‘2011-03-10 23:59’. The second iteration will begin at ‘‘2011-03-11 0:00’. A loop variable entitled CurrentDate is set to the present loop time. Inside this loop, the code runs the iHistorian OPENQUERY SQL code discussed previously substituting the time and tag variables to get the raw SCADA data and insert it into the SQLEXPRESS table. The CurrentDate variable is incremented to the next day, adding one day to the present value. This greatly improves the amount of time required to import data. The full code Matlab code listing is listed in the appendix. It should be noted that importing data into another database is an extra step for offline processing and testing. Creating a permanent solution, if desired, could pull data directly from the SCADA system for processing.

This research included a total of 183 tags out of the 850 available in the historical archive. The tags not chosen include information specific to the treatment process at the pumping stations and treatment plants and not related to the distribution. For instance, there are tags for the running time of pumps and the flow and run time through various filters. For these 183 tags, there are a total of 76,150,080 rows of data, each equating to a single data point observation. So, there are over 416,000 observations for each tag! This data includes observations for approximately 8 months of sampling. The data was collected in the same date range from multiple years to be consistent with seasonal patterns.

Storing this data requires approximately 17.6 GB of disk capacity in the SQL Server Express database. This does not include other relevant information including the tags, pressure zone data, calculated PCA data, and required system SQL Server tables. The SQL Server Express instance also has storage limitations since it is a free product. A single database is limited to 10 GB of space including the database file and associate logs. For this reason, two databases were created in the SQLEXPRESS instance solely for the storage of this raw data. The databases are named RAW0 and RAW1. Each database contains four tables. Database RAW0 contains tables named raw0, raw1, raw2, and raw3. Database RAW1 contains tables named raw4, raw5, raw6, and raw7. Each table has the same structure as the SCADA ihRawData table and contains three hours of data. For example, table raw0 contains all data with times ranging from 0:00 to 2:59 and table raw1 contains all data with times ranging from 3:00 to 5:59. The data was grouped by time because of the methodology for analyzing the data as described further in the paper. Data was also distributed among several tables for improved performance. Each of these tables contains approximately 9.5 million rows or records. Functions to analyze the data and compute the principal components perform table searches based on time. If all data was in one or two tables, the computing process would take an extremely long time since the number of records to analyze would be double or more in size.

PRINCIPAL COMPONENT CALCULATION B

With the data loaded into the SQL Server Express databases, the principal components can be calculated. Matlab code is executed to read through the raw data, grouping the data by hour and minute, and storing the result in another table named HistoricalPCA for processing. This table contains the coefficients or eigenvectors from the principal component analysis.

Start and stop times are set as variables StartTime and EndTime. With the GUI environment, these variables are set through a Matlab listbox. Four listbox components are used for the start hour, start minute, stop hour, and stop minute.

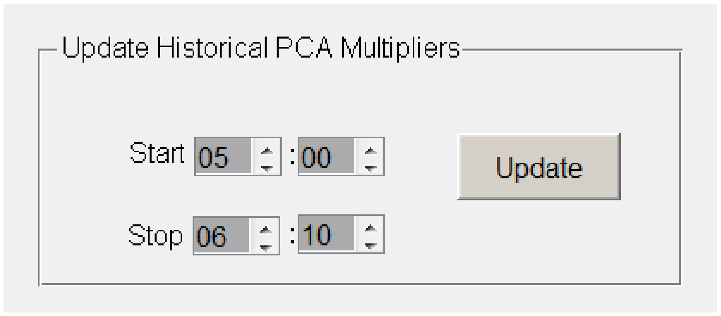


FIGURE 6 - Update Historical PCA Multipliers GUI

Each listbox item has two functions. One function creates the listbox items including the selection for minute or hour options as follows.

```
function stop_min_listbox_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

set(hObject, 'String',
{'01','02','03','04','05','10','15','20','24','25','30','35','40','45','50','55','59'});
```

The other function is the callback function. Each of the callback functions set a global variable to the current, selected value in the listbox control.

```
function stop_min_listbox_Callback(hObject, eventdata, handles)
global stop_min_lb;
stop_min_lb_selected = get(hObject, 'Value');
stop_min_lb_list = get(hObject, 'String');
stop_min_lb = stop_min_lb_list(stop_min_lb_selected);
```

When the Update button is selected in the GUI, it calls a callback function for the pushbutton. The global variables, start_hour_lb, start_min_lb, stop_hour_lb, and stop_min_lb, are used as each contains the string for the start or stop hour and minute values from the listboxes. The start strings are concatenated with a colon to create a start time in HH:MM format and saved to variable StartTime. The stop strings are concatenated in the same manner and saved to variable EndTime. The GUI code also checks that the stop time is not earlier than the starting time. If it is not, it calls a user-defined function to perform the historical PCA analysis. If the stop time is actually earlier than the start time, a message box is displayed and the routine ends so the user can reenter appropriate values.

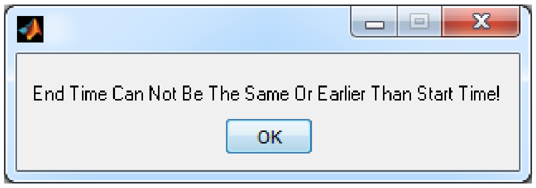

End Time Can Not Be The Same Or Earlier Than Start Time!

FIGURE 7 - Message Box When Incorrect Time Data Entered

The Matlab code executed for the Update pushbutton consists of the following.

```
function upd_hist_pca_pushbutton_Callback(hObject, eventdata, handles)
cla;
global start_hour_lb;
global start_min_lb;
global stop_hour_lb;
global stop_min_lb;

StartTime=strcat(start_hour_lb,':',start_min_lb);
EndTime=strcat(stop_hour_lb',':',stop_min_lb);

StartTimeNum = datenum(StartTime);
EndTimeNum = datenum(EndTime);

if EndTimeNum <= StartTimeNum
   msgbox('End Time Can Not Be The Same Or Earlier Than Start Time!');
else
   update_historical_pca();
end;
```

Once the start and stop times are correctly defined through the listbox routines, the function update_historical_pca begins by setting variables and opening log files for operational management. This function also uses the global start and stop time variables explained previously. Variable alerts_exist is initialized to 0. This variable will be set to 1 when an alert occurs during processing that will get logged to a file. Two files are opened for logging including alerts.txt and log.txt. These files are appended with information. File log.txt contains the start and stop time of processing so the user can see how long the process takes. It also contains information on the time range selected for processing. An example selection is:

```
Update PCA Multiplier Start Run: 2011-10-16 16:53
Update PCA Multiplier Range: 02:22_to_02:24
Update PCA Multiplier Stop Run: 2011-10-16 17:03
```

File alerts.txt contains errors or issues for user awareness. It includes the date and time of the alert and the actual alert. The following example shows several calculations where the data computation included a Matlab Not-A-Number (NaN) result.

```
2011-10-16 17:03: Tag #179 [CHFP.ZONE_DISCHARGE_PRESSURE.F_CV] is NaN
2011-10-16 17:03: Tag #178 [CHFP.ZONE_DISCHARGE_FLOW.F_CV] is NaN
2011-10-16 17:03: Tag #136 [CHFP.SMYRNA_BARDSTOWN_PH.F_CV] is NaN
```

The function also defines a Matlab waitbar which is a timer showing processing progress. This provides the operator a visual on how much longer the routine will take to calculate the PCA components and update system tables. For the number of repetitions to illustrate, it calculates the amount of time between the start and stop times. During each iteration, a step variable is increased which is used for the display.

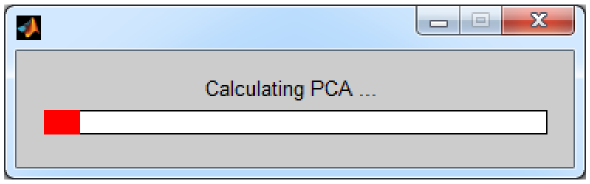


FIGURE 8 - Progress-Bar Illustration

A connection is established to the local SQL Server Express instance SQLEXPRESS for data gathering and processing. Initially, the tagnames are queried, sorted by tagname, and loaded into an array named TagList. The number of tags is computed with the Matlab size command and saved as variable NumTags.

Another string variable is created titled TagString. This variable contains all of the tags used for processing. The string is defined with an opening parenthesis and each tag is enclosed in single quotes. The code takes into account whether or not the tag is the first in the list so it does not contain a comma in the list. An example TagString is (‘CHFP.JEFF_DISCHARGE_FLOW.F_CV’,’CHFP.BROOKS_DISCHARGE_PRESSURE.F_CV’,’CHFP.MARTIN_STATION_TANK_LEVEL.F_CV’).

Next is a loop that begins with the start time and increments until the time reaches the stop time. During the loop, because the data is spread among several tables in the SQL Express instance, the source database and source table are set based on the loop time. For example, at time 02:13, the source database is RAW0 and the source table is RAW0. Another query is executed against the correct database and table to fetch all data for tags included in the TagString variable. The query is further restricted by just the current loop hour and minute, such as 02:13, as well as a prior 4 month date range. This range was chosen to provide a good number of sample points as well as being considerate of weather changes and patterns. If a larger date range was chosen, a broader range of numbers would be returned because of the change in water usage and consumption dealing with weather patterns. Typically, there is higher consumption in summer months compared to winter by homeowners. Including July and August data when evaluating January numbers could have an impact on results. The code for the query is

```
strcat('select * from [',SourceDatabase,'].[dbo].[',SourceTable,'] where timestamp >=
"2009-03-10 00:02" and timestamp <= "2009-07-03 23:59" and tagname in ',TagString,'
and convert(time,timestamp) = ''',datestr(StartTimeNum,'HH:MM'),''' order by timestamp
asc').
```

This illustrates the variables being used. With variable substitution, this becomes

```
select * from [RAW0].[dbo].[RAW0] where timestamp >= "2009-03-10 00:02" and
timestamp <= "2009-07-03 23:59" and tagname in
('CHFP.JEFF_DISCHARGE_FLOW.F_CV','CHFP.BROOKS_DISCHARGE_PRESSU
RE.F_CV','CHFP.MARTIN_STATION_TANK_LEVEL.F_CV') and
convert(time,timestamp) = '2:13' order by timestamp asc').
```

The result of this SQL query is stored in variable bigdata. This contains all fields including tag name, date timestamp, raw data value, and quality for all tags at the loop time hour and minute. There could be issues with the SCADA raw data due to maintenance windows causing missing readings, new devices installed, or old monitoring points removed. This could make some tags have more rows of data returned than others. For this reason, another loop is executed to find the highest number of rows returned that all tags have in common. This allows all of the tag data to have the same number of samples to examine and can be easily used with Matlab matrix operations. If the number of rows were different, the matrix operations would fail. As it loops through the data in the bigdata variable, it uses the Matlab functions find and ismember to query the set and only respond to the current tag in the loop. The Matlab size function then gathers the number of rows returned for that specific tag. If the value is less than the current minimum, the new minimum becomes this size. The result of this internal loop is a variable called minrows that stores the number of rows that each data point has in common.

```
minrows = 32000;
   for i = 1:NumTags
```

```
    cursize = size(find(ismember(bigdata,TagList(i))),1);
    if cursize < minrows
        minrows = cursize;
    end
end
```

During each iteration of the main loop, an empty matrix H is initialized. This will store the raw historical data. Each column of this matrix will contain the raw data and have minrows of data from the previous calculation. Similar to the previous operation, another internal loop through all of the tags is utilized. The find and ismember Matlab functions are used again to find all raw data for the current tag, returning arrays r and c containing the row and column information locating the current tag data in the bigdata matrix. All rows for the current tag are sorted ascending by time which is the second column of the bigdata matrix. The actual raw data, which is the third column of the bigdata matrix, is converted from a string to a double and only includes the last minrows number of data values. This data is concatenated to the H matrix with the Matlab horzcat function. Once all of the tags have been processed, matrix H contains all of the historical raw data. Each column corresponds to a specific tag. Since all tags were searched and processed in alphabetical order, it is important to note that column one of H corresponds to the first tag processed and the nth column of H corresponds to the nth tag processed. This will make future calculations easier by matching the tag and raw data indexes.

As stated previously, the raw data can contain variations in the data including null or 0 data. For instance, if a storage tank is out of service for maintenance, the SCADA system may return null or 0 values during that time period. There are several methods to deal with missing data. The most usual way is to completely delete any observation for which at least one of the variables has a missing value [7]. This method is used further in the PCA calculation. Another method is to replace missing values with the mean calculated from observations for which the value is available [7]. For the data in matrix H, all zero data is replaced by the mean of the non-zero elements to minimize data loss. Matlab functions are used to loop through a temporary copy of matrix H, searching for zero values. When located, their row and column values are saved in temporary array variables. Looping through each column, if a zero value is discovered, the data point is deleted and leaves only non-zero elements in the array. Another array named averages is used to store the average of the remaining non-zero elements. The size of array averages equals the number of columns in H which is the number of SCADA tags being used. Another loop is used to set the zero values in H to the average.

With all zero values removed, the data from H is normalized into a new matrix called Hn. For this operation, the number of rows and columns of H are required and stored in variables histn and histm respectively. The mean of each column is computed and stored in variable HMean. The standard deviation of each column is calculated with the Matlab std function and saved to variable HStd. These values are used in the normalization data calculation. The normalization is calculated by subtracting the mean from each data point and dividing by the standard deviation.

```
% n = number of rows
% m = number of columns (ie variables)
[histn histm] = size(H);

% Calculate mean for each column
HMean = mean(H);

% Calculate Standard Deviation for each column
HStd = std(H);

% Calculate normalized data
Hn = (H - repmat(HMean,[histn 1])) ./ repmat(HStd,[histn 1]);
```

With a new matrix Hn containing the normalized raw data, there may still be data integrity issues. In the previous calculations, all zero data was replaced with the average of non-zero data. What happens if all data during a timeframe is zero? This data point must be removed entirely as was explained the primary option when dealing with missing data. In the normalized matrix, zero or missing data is discovered by NaN values indicating Not-A-Number. These are instances where the mathematical operations to calculate the normalized data have undefined results. The Matlab isnan function locates these values. To prevent removing a tag or variable from all further processing, temporary tag lists and tag counts are used. This way, further processing still uses all variables if data is available. If a NaN value exists, the event is logged in the alert log and the alerts_exist variable is set to 1 to signify the existence of an issue. The value in the HistoricalPCA result table for the tag with NaN is set to 0 for the current hour and minute. This is similar to deleting the value since anything multiplied by it will be 0 and adding anything to it will also be 0. The tag is removed from the temporary tag list variable and the count of tags is reduced by one. The normalized data column is also removed from Hn completely removing the data from the PCA calculation. It is interesting to note that the loop to remove NaN columns starts with the last entry and ends with the first data point. Calculations are done in this manner because of the indexing that is used to keep the tag and data values in sync with regards to matrix operations. If the loop was started at one, and the second column needed to be removed, the second column data would be removed and the variables modified as needed. However, when the column is removed from the matrix, the third column becomes the second column and so-on. The next iteration of the loop would be the third column but by comparing the original matrix, this is actually the fourth column! By starting with the last tag, there is no adjusting of columns. For example, if there are 12 variables resulting in 12 columns in the Hn matrix, and column 11 is removed, column 12 still becomes column 11. However, since working from end to beginning, the next iteration would be column 10 as expected. No adjustments are needed to keep the indexing in sync. Removing the columns with NaN values is completed with the following Matlab code.

```
% Check for NaN in matrix due to non-changing variables
% Set HistoricalPCA value to 0 for future processing to not use it and
% remove the tag from current processing
tTagList = TagList;
tNumTags = NumTags;
j = NumTags;
while j >= 1
   if isnan(Hn(:,j))
      fprintf(alert_fid,'%s\n',(strcat(datestr(now,'yyyy-mm-dd HH:MM'),': Tag
#',num2str(j),' [',cell2mat(tTagList(j)),'] is NaN')));
      alerts_exist = 1;
      %update value with 0 in HistoricalPCA table
      sqlquery=strcat('update [scada].[dbo].[HistoricalPCA] set [',tTagList(j),'] = 0 where
Time = ''',datestr(StartTimeNum,'HH:MM'),''';');
      curs = exec(conn,sqlquery);

      %remove Hn and TagList and subtract 1 from NumTags
      tTagList(j)=[];
```

```
        Hn(:,j)=[];
        tNumTags = tNumTags - 1;
    end
    j = j - 1;
end
```

After all of these calculations and preparation, the principal components can finally be determined! The Matlab princomp function is used on the Hn normalized matrix.

```
% Perform PCA on normalized data
[HCOEFF HSCORE HLATENT] = princomp(Hn);
```

The princomp function returns three variables. HCOEFF stores the unit eigenvectors, HSCORE stores the PCA results, and HLATENT contains the eigenvalues. The eigenvectors are the data stored in the HistoricalPCA data. These values will be used in future processing as coefficients to calculate a current PCA value. Another internal loop processes each remaining tag and updates the current hour and minute of the HistoricalPCA table with its corresponding HCOEFF value.

```
% store HCOEFF into Historical table
for i = 1:tNumTags
    sqlquery=strcat('update [scada].[dbo].[HistoricalPCA] set [',tTagList(i),'] =
',num2str(HCOEFF(i,1),16),' where Time = ''',datestr(StartTimeNum,'HH:MM'),''';');
    curs = exec(conn,sqlquery);
end
```

An example update for a non-zero data point would be 'update [scada].[dbo].[HistoricalPCA] set ['CHFP.BROOKS_STATION_TANK_LEVEL.F_CV'] = -0.0379695589580 where Time = '02:13';'

Several checks are computed on the results to ensure accuracy and to maintain faith in the system. The first checks that the sum of the eigenvalues equals the number of variables or tags. In using all of the tags, they both equaled 182. The second check ensured the sum of the variance of the computed principal components equaled the number of variables or tags. Again, using all tags, they both equaled 182. In either case, if the checks did not equal the correct values, it is written to the alert log.

```
% perform checks on HLATENT and manually calculate variance on HCOEFF
str1 = sprintf('%.4f',sum(HLATENT));
str2 = sprintf('%.4f',histm);
str3 = sprintf('%.4f',sum(var(HSCORE)));
if str1 ~= str2
   fprintf(alert_fid,'%s\n',(strcat(datestr(now,'yyyy-mm-dd
HH:MM'),':',datestr(StartTimeNum,'HH:MM'),': HLATENT sum != number of
columns!')));
   alerts_exist = 1;
   %disp('HLATENT sum != number of columns!');
end

if str3 ~= str2
   fprintf(alert_fid,'%s\n',(strcat(datestr(now,'yyyy-mm-dd
HH:MM'),':',datestr(StartTimeNum,'HH:MM'),': HSCORE variance sum != number of
columns!')));
   alerts_exist = 1;
   %disp('HSCORE variance sum != number of columns!');
end
```

All of these computations build the HistoricalPCA table in the local SCADA database. It has the following format.

TABLE VIII

[SCADA].[dbo].[HistoricalPCA] Table Structure

COLUMN_NAME	DATA_TYPE
Time	Time(7)
[CHFP.BROOKS_STATION_TANK_LEVEL.F_CV]	Decimal(18,13)
[CHFP.JEFF_FOREST_TANK_LEVEL.F_CV]	Decimal(18,13)
[CHFP.MARTIN_STATION_TANK_LEVEL.F_CV]	Decimal(18,13)
Repeats for rest of SCADA tags	Decimal(18,13)

Because each column is also a SCADA tag, there are a total of 183 columns counting the Time field and the 182 tags. The Time field contains unique values from

0:00 to 23:59 for each minute of the day. Therefore, there are 1,440 rows in the table. The data in each tag's field has the calculated PCA coefficients for a specific time. For instance, as shown in the following sample, at time 0:01, the PCA coefficient for the CHFP.BROOKS_STATION_TANK_LEVEL.F_CV is -0.0508022294967.

TABLE IX

[SCADA].[dbo].[HistoricalPCA] Single Field Sample

Time	CHFP.BROOKS_STATION_TANK_LEVEL.F_CV
0:00	-0.0503698239275
0:01	-0.0508022294967
0:02	-0.0536926964676
0:03	-0.0527105587950
0:04	-0.0516202908944

Initial designs of this process included additional SQL queries to the local SQLEXPRESS database for information gathering. Specifically, the code that discovers the highest common count of data among all tags had many more SQL select queries. At first, there was one select statement to get the highest common count and then a loop for all tags to return that amount of rows of data. When dealing with a small pressure zone or area with a small amount of tags, the process ran smoothly. When running this process with 182 tags, it took quite a while to run. For each minute to populate the HistoricalPCA table, there was actually 18 minutes of processing time. For all 1,440 rows in the table, it would take 25,920 minutes or 18 days to calculate everything. The updated code presented takes advantage of Matlab matrix processing. Instead of performing a SQL select for each tag to parse the correct number of data samples, a single SQL select is executed with the results stored in a matrix. Matlab ismember and other commands explained previously then parse the correct number of samples. This method reduces the amount of processing time to 3 minutes per HistoricalPCA row.

PRINCIPAL COMPONENT OPERATION C

Once the HistoricalPCA table holds the PCA coefficient calculations, analysis can be made on other SCADA data to look for anomalies and potential breaks or issues in the distribution system. Matlab is again used to query the SQLEXPRESS database at the current time and use the coefficients from the HistoricalPCA table to create a current principal component value. The variance of the current principal component value and the prior 20 minutes values is calculated as well. The difference between the calculations at each minute's sampling is used to show the possibility of a system issue. Both the current principal component value and associated variance are stored in a results table. This table is named the same as the pressure zone that is being evaluated. For all tags, which are the primary focus of this research, the table name is PZALL. This table and any others storing the calculated data, has the following structure.

TABLE X

[SCADA].[dbo].[PZALL] Table Structure

COLUMN_NAME	DATA_TYPE
TimeStamp	Datetime
CurrentValue	Decimal(28,14)
Variance	Decimal(28,14)

For research purposes, the current time was simulated and another table similar to ihRawdata is used. This table contains the same SCADA tags and is new raw data that was not used in the calculation of the HistoricalPCA coefficients. The Matlab code initializes several variables including the beginning time, ending time, and what pressure zone is being analyzed. For this research, it was predominately set to PZALL indicating all tags from all pressure zones would be analyzed. This also corresponds to the table that will store the final results as explained previously. A connection to the local SQLEXPRESS database is also created.

The list of tags to be used in the operation is created first. The program queries the PRESSURE_ZONES table for all required tags. The results are stored in array TagList with the number of tags returned in variable NumTags. This portion of code is listed below. The results are sorted by the tagname field along with future queries. This allows the TagList to be referenced by an index and the index remains the same throughout the other queries. For instance, TagList(1) references tag CHFP.BROOKS_STATION_TANK_LEVEL.F_CV and TagList(2) references tag CHFP.JEFF_FOREST_TANK_LEVEL.F_CV. In future queries, when needing to use one of these tags, it can be referenced by either TagList(1) or TagList(2) instead of doing another lookup into the database.

```
% Get the tags from the pressure zone
sqlquery = strcat('select tagname from [scada].[dbo].[pressure_zones] where
PressureZone = ''',PZTable,''' order by tagname;');
curs = exec(conn,sqlquery);
curs = fetch(curs);
TagList = curs.data;
NumTags = size(curs.data,1);
```

Similar to previous components, string variables named TagString and TagsHistString are created. Each contains a comma separated list of the tags that will be used in future SQL queries. The TagString is enclosed in parenthesis. An example is ('CHFP.BROOKS_STATION_TANK_LEVEL.F_CV','CHFP.JEFF_FOREST_TANK_LEVEL.F_CV','CHFP.MARTIN_STATION_TANK_LEVEL.F_CV'). The TagHistString is another, similar string that will be used to query the HistoricalPCA table. Since the tags are the actual field names, each tag needs to be enclosed in square brackets. An example TagHistString is [CHFP.BROOKS_STATION_TANK_LEVEL.F_CV],[CHFP.JEFF_FOREST_TANK_LEVEL.F_CV],[CHFP.MARTIN_STATION_TANK_LEVEL.F_CV].

Next, the entire HistoricalPCA table is queried and loaded into a Matlab matrix with name HISTPCA. This was chosen as a way to speed processing. Instead of having to run a select query for each minute, this is only computed once and stored in Matlab memory. The Time field in the table was converted to VARCHAR format so it could be queried in the HISTPCA matrix itself. The TagHistString was used in the query to only return the fields or tags that we are interested in processing. This portion of code includes the following.

```
sqlquery = strcat('select CONVERT(varchar,time,108) as thetime,',TagHistString,' from
[scada].[dbo].[HistoricalPCA] order by thetime;');
curs = exec(conn,sqlquery);
curs = fetch(curs);
HISTPCA = curs.data;
```

A loop is then executed starting with the start time provided and ending with the end time provided. A variable CurrentTotal is initialized to zero. This variable will contain the computed principal component for the current date / time. The current raw data is then queried with the statements below. It only queries the tags included in TagString. As explained previously, it also sorts the results by the tagname field to keep the same order as the original queries. This allows the use of the indexed TagList array.

```
sqlquery = strcat('select * from [ihist].[dbo].[ihRawdata] where tagname in ',TagString,'
and timestamp ='",datestr(BeginDateNum,'yyyy-mm-dd HH:MM'),"' order by tagname;');
curs = exec(conn,sqlquery);
curs = fetch(curs);
curdata = curs.data;
```

The current raw data values are then multiplied by the historically calculated PCA coefficients stored in the HISTPCA matrix to determine the principal component. The current hour is stored as variable curHH and the current minute is stored at curMM. These two variables are used to determine the correct row from the HISTPCA matrix. Since it is a matrix ordered on time, the row that matches the current hour and minute can be calculated by adding the product of the hour and 60 with the minutes and adding another one. Basically it computes what minute of the day it is based on 1440 minutes total in the day. For example, 0:00 is obviously the first row of the HISTPCA matrix. Multiplying the hour 0 times 60, adding the minutes 0, and then adding another 1 yields 1, which is the first row. For time 2:13, row 134 is selected since 2 times 60 plus 13 plus 1 equals 134. The array variable histrow stores the values from the HISTPCA matrix. It should be noted that histrow contains data from columns two through the number of tags stored in the NumTags variable plus one. The first column contains the time so it is removed.

The CurrentTotal variable is then calculated through matrix multiplication. The historical coefficients for the current time stored in variable histrow are multiplied by the current raw data. The matrices will have the same dimensions because all of the queries and resulting variables were limited and ordered by the appropriate tags. Also, only the third column of the returned raw data is used which is the raw data point. By multiplying the matrix with the array, it automatically sums the data which becomes the principal component. These steps are completed with the following Matlab code.

```
curHH = str2num(datestr(BeginDateNum,'HH'));
curMM = str2num(datestr(BeginDateNum,'MM'));
rownum = (60 * curHH) + curMM + 1;
histrow = cell2mat(HISTPCA(rownum,2:(NumTags + 1)));
CurrentTotal = histrow * str2double(curdata(:,3));
```

Now that the current principal component has been determined, the variance in the data needs to be calculated and the results stored in the corresponding table. The variance is calculated with the prior 20 minutes data. For this, a query is executed on the results table for data greater than 20 minutes prior. Since it is greater than and not greater than or equal to, it actually returns the previous 19 minutes of data. This is ok since the current value will be in the list as well giving 20 minutes of data. The query results will be returned in a Matlab cell format. For mathematical calculations, this is converted to number format and saved in array TheCV for the current value. The current value is appended to the array with Matlab array operations. The variance is then computed using the Matlab var function and saved in variable TheVar. The following code demonstrates this operation.

```
sqlquery = strcat('select currentvalue from [scada].[dbo].[',PZTable,'] where timestamp
<=''',datestr(BeginDateNum,'yyyy-mm-dd HH:MM'),''' and timestamp
>''',datestr(addtodate(BeginDateNum,-20,'minute'),'yyyy-mm-dd HH:MM'),''' order by
TimeStamp;');
curs = exec(conn,sqlquery);
curs = fetch(curs);
TheCV = cell2mat(curs.data);
TheCV = [ TheCV ; CurrentTotal ]; % Add current value to vector since not in table yet
TheVar = var(TheCV);
```

This data is inserted into the results table with the Matlab code below.

```
sqlquery = strcat('insert into
[scada].[dbo].[',PZTable,'](TimeStamp,CurrentValue,Variance) values
(''',datestr(BeginDateNum,'yyyy-mm-dd
HH:MM'),''',',num2str(CurrentTotal),',',num2str(TheVar),');');
curs = exec(conn,sqlquery);
```

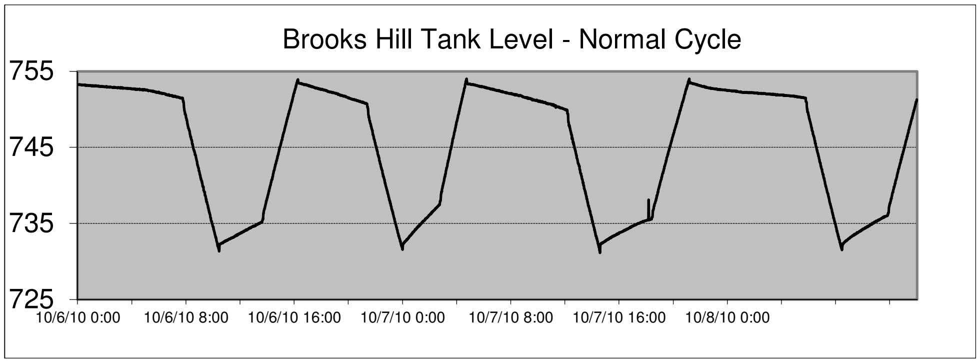
The current date and time is incremented by one minute and the process is repeated. The results from this stage are storing the principal component and the twenty minute variance for plotting and analysis.

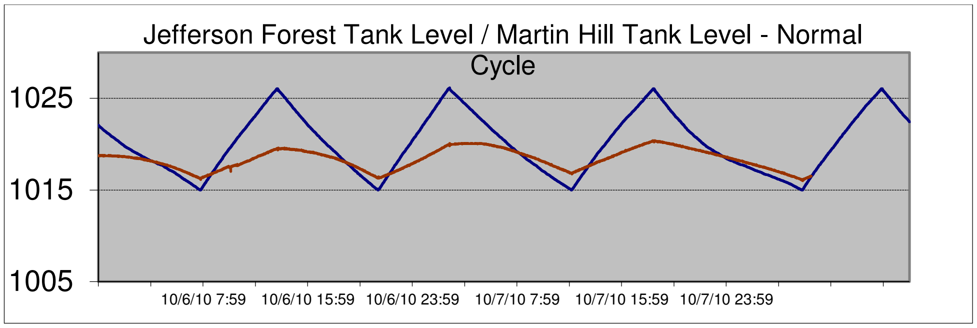
V. RESULTS

INITIAL TESTING A

To verify the process, this research started by evaluating data in a small pressure zone. This pressure zone is one of the zones in the 760 pressure plane named Brooks Hill and has four SCADA tags. Working with the SCADA administrator, the date and information concerning a past incident was provided. This would allow the logic and mathematical analysis to be tested in a small, closed environment. Since it only contained four tags, visual analysis of the raw data can show the issue by comparing to normal operations.

The SCADA tags in this simple environment consisted of CHFP.BROOKS_STATION_TANK_LEVEL.F_CV, CHFP.JEFF_DISCHARGE_PRESSURE.F_CV, CHFP.JEFF_FOREST_TANK_LEVEL.F_CV, and CHFP.MARTIN_STATION_TANK_LEVEL.F_CV. Charting the raw data under normal circumstances provided the following information.





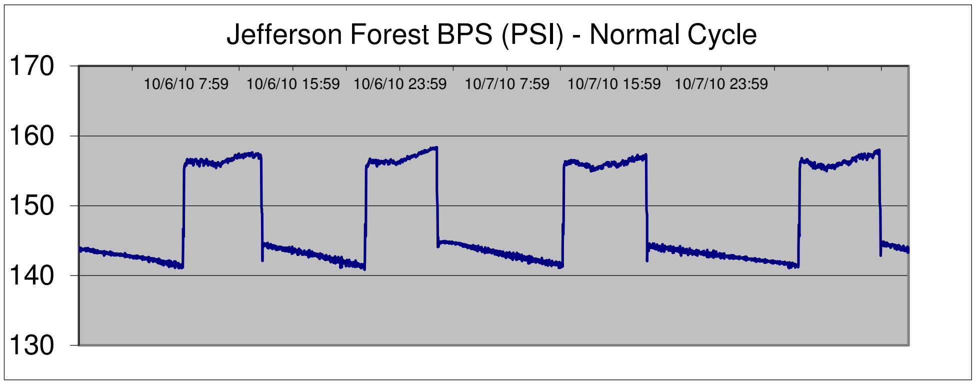
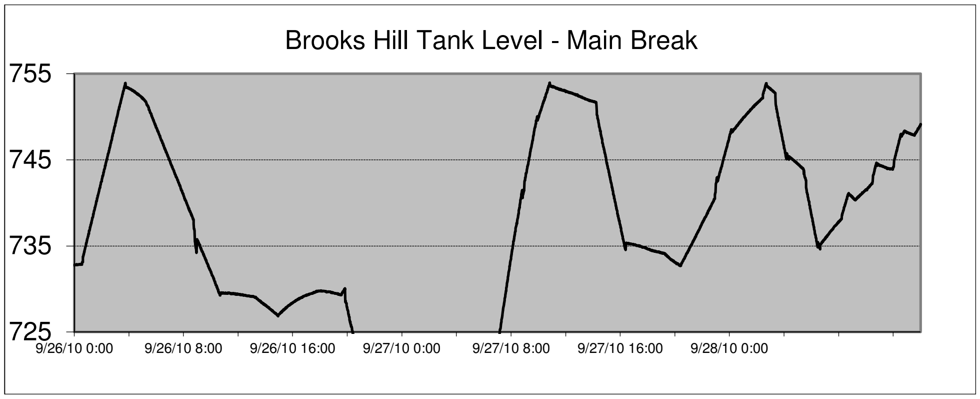
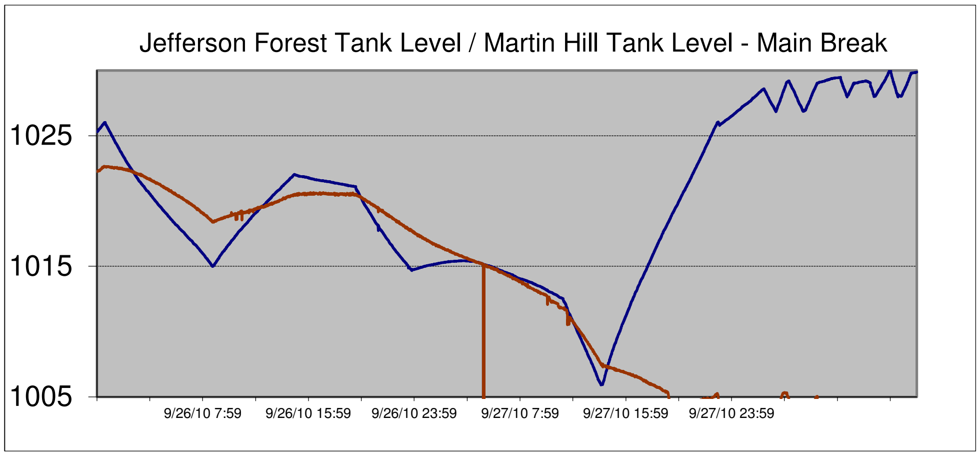


FIGURE 9 - Brooks Hill 760 Pressure Zone Normal Operations

Over a several day span, it is easy to notice the normal data pattern representing the typical usage. Charting the raw data during a break illustrates a drastic change in this pattern.





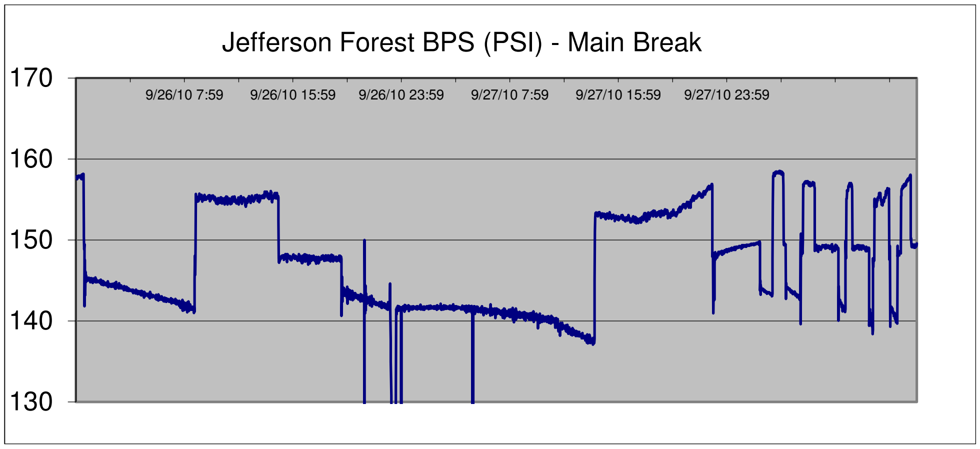


FIGURE 10 - Brooks Hill 760 Pressure Zone Break Condition

It is obvious that a break condition was occurring by reviewing the drastic changes in raw data. The same raw data used in these charts were processed with the PCA programming outlined previously. This small group of variables or tags was placed in a pressure zone named ORIG4. The PressureZone name in the PRESSURE_ZONES table was set to ORIG4 for these tags. The output table containing the principal components and calculated variance is named ORIG4. Using the difference between each 20 minute variance samples produced the following results.

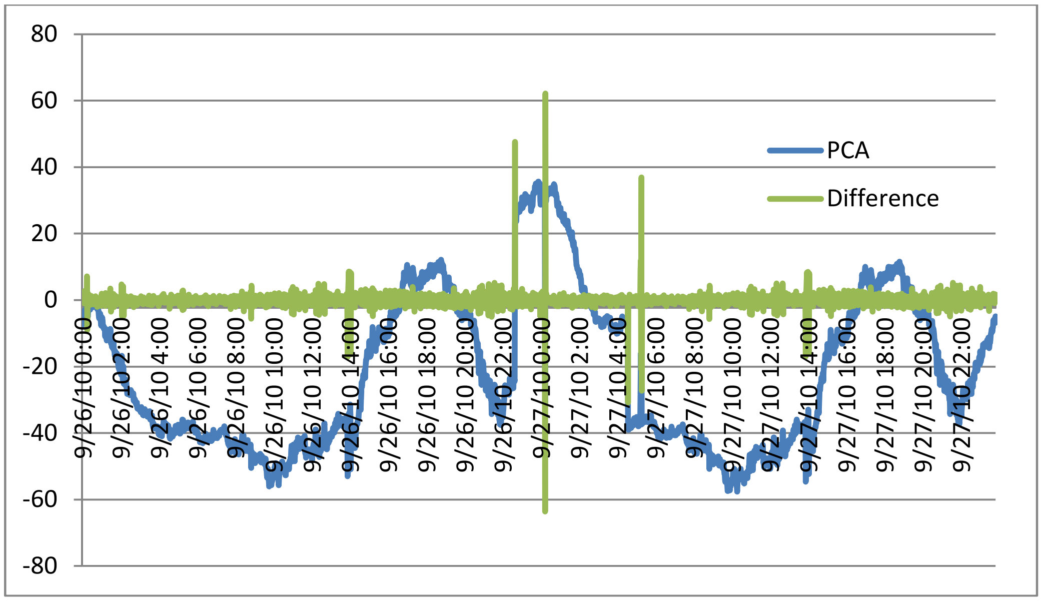


FIGURE 11 - PCA Calculation Example using 20 Minute Variance

The chart above was created in Microsoft Excel after importing the ORIG4 tabular data from the SQLSERVER database instance. Both lines show variations in data. The blue line represents the calculated PCA for the small testing area. The green line represents the difference between the current calculated variance and the variance from the previous minute. Again, the calculated variance included the current and prior 19 minute PCA computations. One can also notice a pattern in the PCA line just as a pattern is noticeable in the raw data due to usage demands and pumping trends.

It is interesting to match the break information of the PCA computed data with the raw SCADA information. From viewing the raw data chart, it appears that the issue occurred between 14:00 and 16:00 and had major issues around midnight. The PCA calculated graph shows a change in variance shortly before 14:00 and major changes just after 22:00 and midnight.

From these observations, the process described to compute the principal component and then compare variance data was accurate. It was then required to determine what affect the 20 minute sampling range had on the difference among variance calculations. Additional ranges were then tested including 15 minute, 45 minute, and 60 minute ranges. These range changes changed the number of previous samples used in the variance calculation. For instance, at time 13:30, the 20 minute chart above included the 20 PCA calculations sampled from 13:11 to 13:30 in determining the variance. Changing the range to 60 minutes, for example, increased the number of samples in the variance calculation to 60 and included samples from 12:31 to 13:30. With the same set of data, this produced the following results.

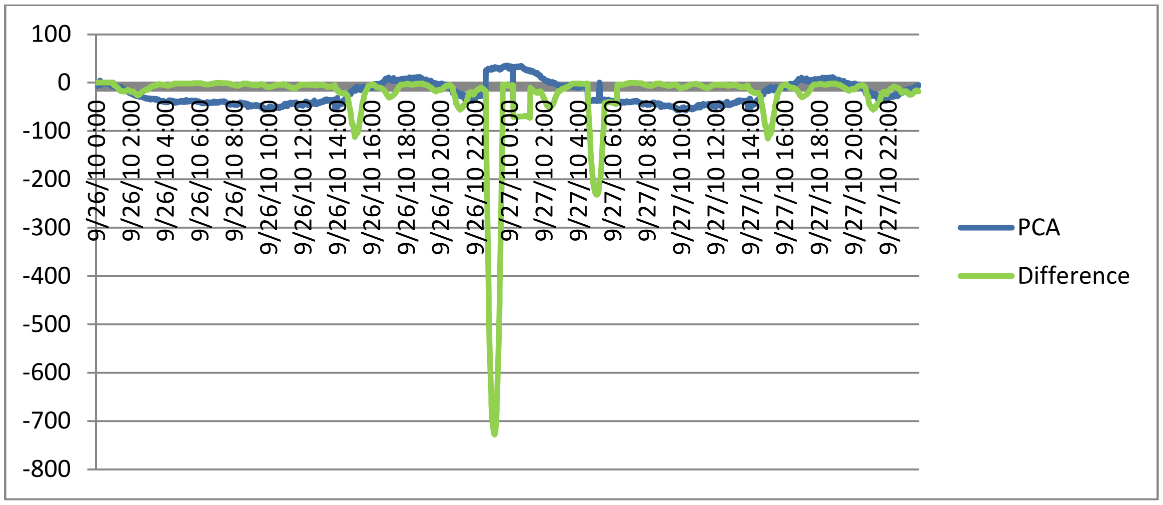


FIGURE 12 - PCA Calculation Example with 60 Minute Variance

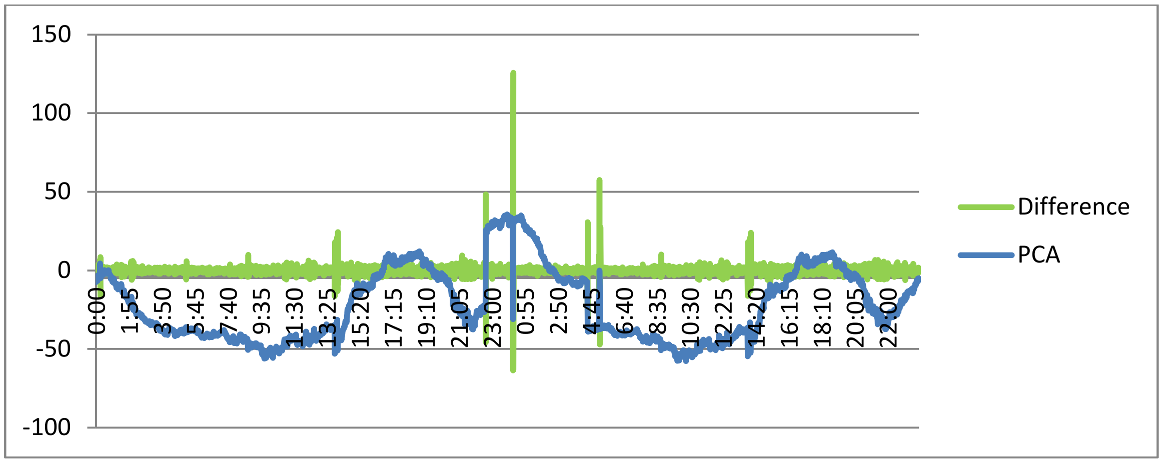


FIGURE 13 - PCA Calculation Example with 45 Minute Variance

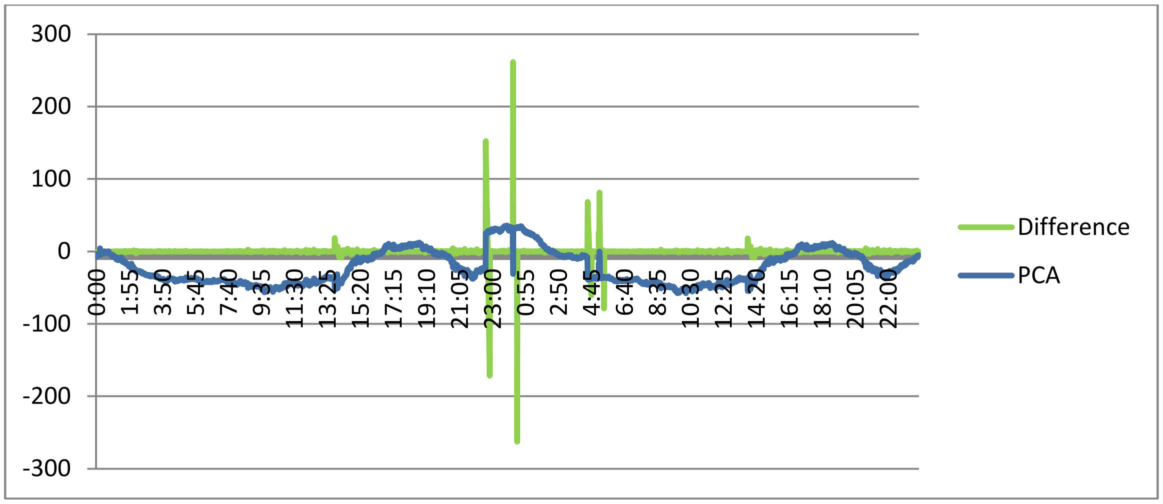


FIGURE 14 - PCA Calculation Example with 15 Minute Variance

Each method shows the same PCA data even though the range on the x-axis has different interval values. They each show the issue occurring at approximately the same time. The 60 minute variance showed a much different graph making it stand out more. Was this the best variance range for detection? To make a more informed decision, a smaller time interval was compared. Specifically, a smaller time range was charted to show more detail. The early warning time of the break seemed to occur around 14:00 so that area was highlighted. Adding all variance ranges on the same chart and therefore the same time axis illustrated the differences and similarities. As perceived from the individual charts, the 60 minute range was the most dramatic. The others maintained a very similar pattern, all around the same time within about 15 minute of each other. The 20 minute range was chosen even though the 60 minute range had the most obvious difference. From an early reporting perspective, the 60 minute range showed the largest change but it wasn’t until much later in time, around 15:00 that it had the highest change which was approximately -100. Both the 15 minute and 20 minute ranges showed the earliest changes at around 13:52. Between these two, the 20 minute seemed to maintain additional, noticeable spikes in the data making it more useful. The 45 minute range was very close to both the 15 and 20 minute ranges but just a few minutes past the others. Trying to create a predictive or early warning application steered the decision towards the

20 minute range although any option would be acceptable. The chart below was created in MS Excel to show these differences.

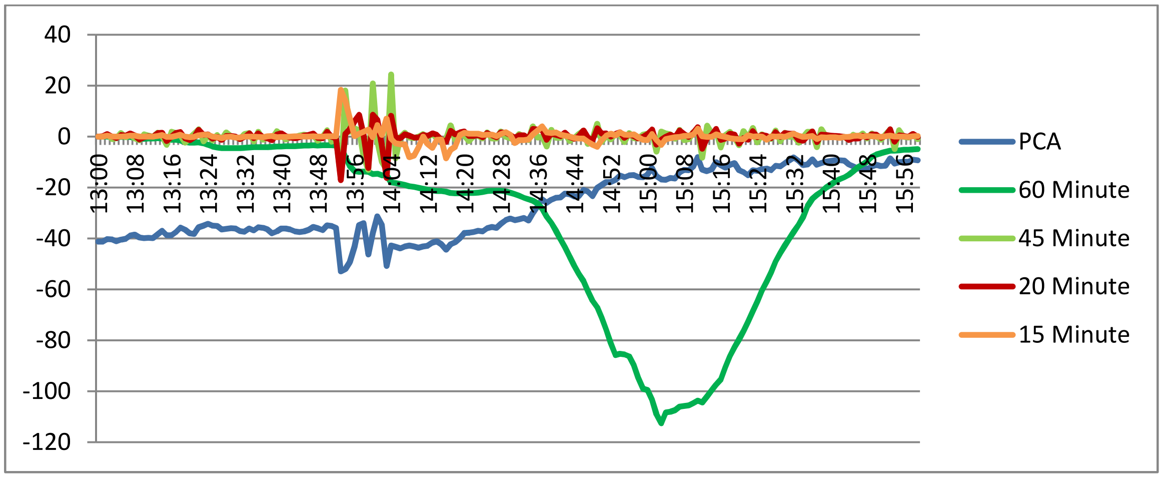


FIGURE 15 - Comparison of Variance Calculations

<u>ENTERPRISE ANALYSIS B</u>

With the process defined and initial testing complete, the next stage of research was to enlarge the testing environment to include the entire distribution system. Instead of using four variables in basically a closed system, 182 variables would be evaluated. Performing the same process yielded similar results but also presented several questions and challenges. One of the first tests included analysis when no major incidents occurred in the system. Every day, Louisville Water Company deals with breaks and other system issues. A time was chosen that no *large* break occurred. The hypothesis was that the change in variance would be minor and a pattern of the PCA data would still exist. The MS Excel chart below shows a normal operating day using the 20 minute range of PCA data for the variance.

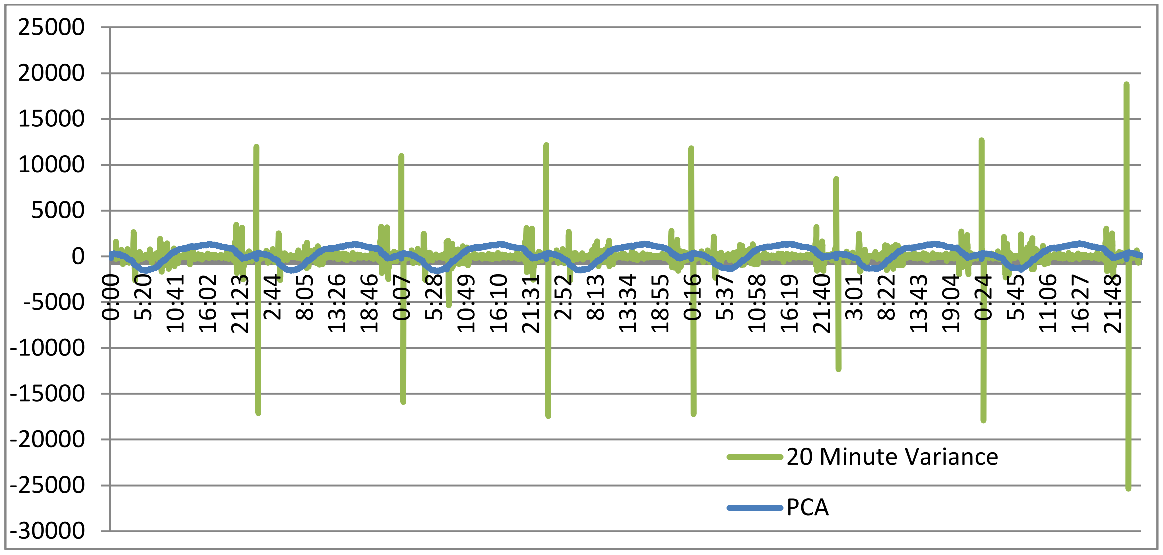


FIGURE 16 - Normal Operating Day

Considering that issues occur each day as well as normal maintenance of pipe, valves, and other system components, some change in variance was expected. The large differences in variance around midnight each day was completely unexpected. The chart above includes data from a seven day span. As originally thought, the PCA data does maintain a repeating pattern when viewed at a high level. This is similar to the repeating pattern of the raw data. The spikes of the variance data also show a repeating pattern although not as exact and defined.

With this new data, there was concern at what the break data would show. Working with the SCADA Administrator, several dates were chosen when large breaks occurred. Similar to the other charts, the break conditions showed the repeating pattern of PCA data and the large spikes around midnight. Below are two different incidents that have taken place in our distribution system.

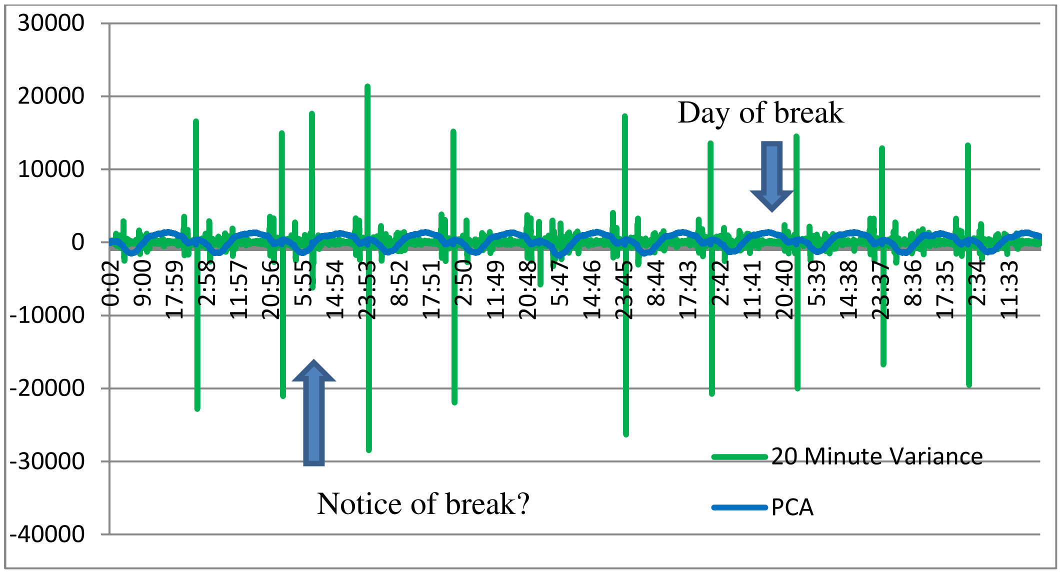


FIGURE 17 - First Large Break Occurrence

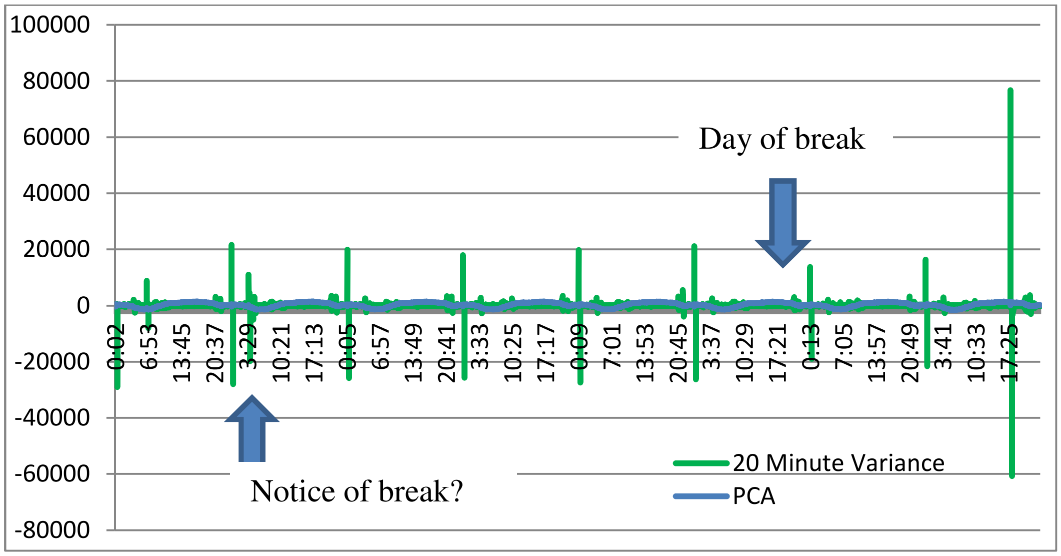


FIGURE 18 - Second Large Break Occurrence

During each testing date range, there were large changes in the variance at different times. Again, there were large changes around midnight. In the second case, there was a large change several days after the incident at approximately 17:25. This was later determined to be the result of making system changes to remove the broken section of main from the distribution network. In the initial testing with the small environment, the PCA calculations and change in difference matched closely to the raw data. There was some advanced warning but not by much. Using 182 SCADA tags, the raw data could not be plotted and compared. That was the point of using PCA! For the large amount of data, there were observances of large changes in variance that occurred several days prior to the notice of the break. In the first case, it was four days prior and the second was five days. It should be noted that the day and time of the break was determined by examining several sources. One, the actual breaking of the pipe causing damage to the ground and surrounding area depending on the severity of the break was an obvious indicator. Secondly, the raw SCADA information was used to notice abnormalities with the data. Once there is a known break in a specific area, the raw SCADA information can be reviewed for tags in that region. This is similar to our evaluation in the initial testing phase with the small number of tags. For these two instances, there was a great pressure drop at one of the SCADA points in the region of these breaks. The first break had a large drop at 18:00 and the second break at 20:00. The charts presented above are very high level showing data from several days and these specific times do not show anything abnormal due to the lack of detail. Graphing the PCA and variance data specifically at the time of the pressure drop resulted in the following.

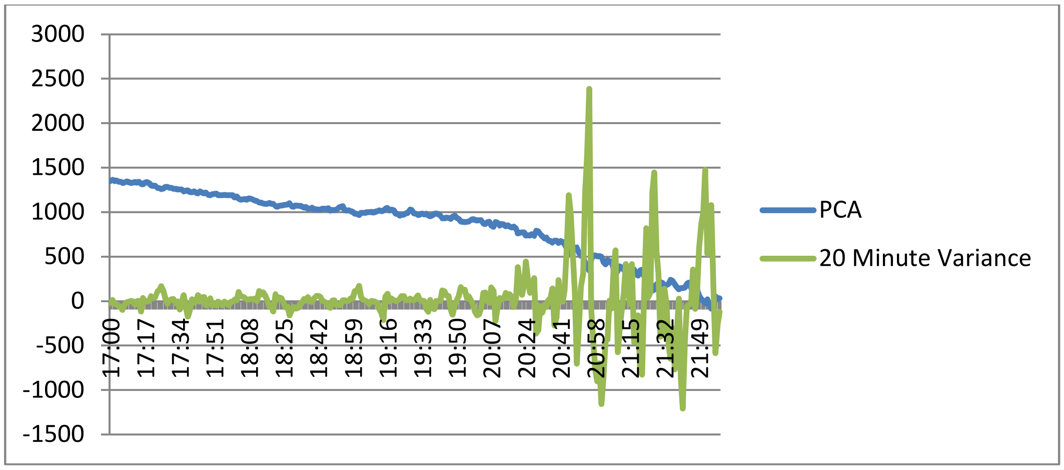


FIGURE 19 - First Large Break Occurrence Detail

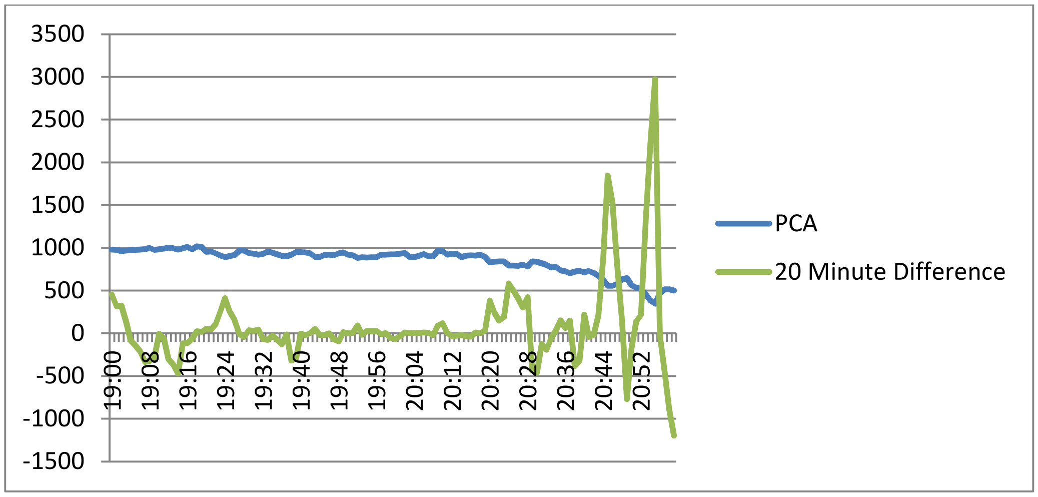


FIGURE 20 - Second Large Break Occurrence Detail

The detail view of the first break occurrence showed a change in variance starting around 20:24. The second break started the variance change around 20:20. The second was closer to the raw SCADA pressure drop information but both did show changes using the PCA calculated data. Relating to earlier discussions of a small test environment, it must be noted that these charts are using 182 SCADA tags instead of one indicating the large pressure change. One variable may or may not have a large impact on the overall PCA calculations depending on how drastic of a change occurs, if any.

In the charts with multiple days of information, a distinct pattern existed especially when reviewing the calculated PCA normal operational data. The variance data also showed a pattern including a regular spike in data at midnight and other times. It was discovered that these patterns of variance change could be the result of operational practices. The regular starting and stopping of pumps as needed by demand can be cyclic in nature, especially when looking at certain times of year. For instance, with higher demand in the summer, additional pumping is required and can be shown by changes in the SCADA PCA results. The pattern in the summer can look different than the winter results due to lower demand. Practicing quality controls to limit excessive electrical usage is believed to be the cause of the large spikes around midnight. The electrical rates are higher in peak demand times during the day. The high rate penalty is lifted around 11 pm allowing LWC to increase pumping for filling storage facilities or provide electricity to other components in the system. Normal maintenance may also cause the change in variance even during regular operations. If a storage tank, pump, or section of pipe is removed from operations for maintenance, it can trigger a change as well [4].

RUNTIME ENVIRONMENT C

To model the above environment on a PC as an end user support system, a Matlab graphical user interface was created. Much of the code was detailed previously on the actual PCA calculations. In particular, the update operation for computing the actual principal component coefficients was explained thoroughly. This is accomplished by selecting the Update button in the user interface. The user needs to be aware that this is a long-running process if choosing a wide span of time. Each minute that it needs to calculate takes three minutes of actual time!

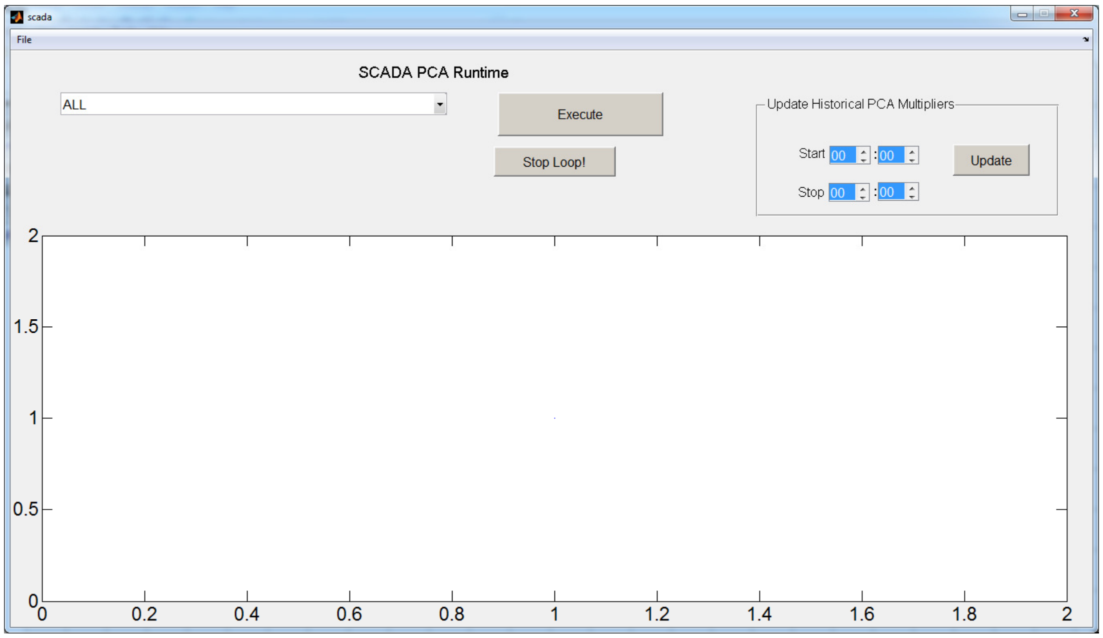


FIGURE 21 - Runtime Interface Starting View

When the application starts, a blank window is displayed. To begin charting the PCA and variance data, the user needs to select an option from a drop-down list. With the original focus to look at pressure zones individually, the drop-down list shows the pressure zones for the system as well as an option to select all data. As a testing environment, only a few pressure zones were included. The small test pressure zone entitled “Pressure Zone 760 – Brooks Hill” and all SCADA tags in our test instance entitled “ALL” were the primary focus.

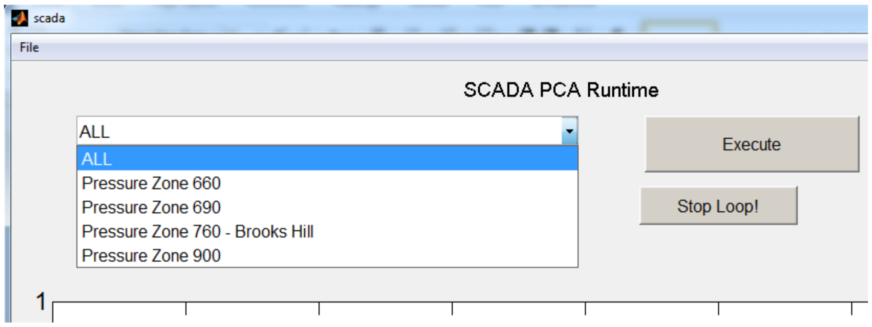


FIGURE 22 - Runtime Pressure Zone Listing

Once a pressure zone is selected, the user can hit the Execute button to begin the plot. In the real-life application, the plot would start with the current time. For research purposes, the start time was hardcoded in the application. With the Execute button pushed, a global variable, emergency_stop is set to 0. This variable is used for control of the plot window in conjunction with the Stop Loop button. The Stop Loop button can be used to halt the drawing in the plot window. The code behind the Stop Loop button simply sets the global emergency_stop variable to 1. While a plot is still drawing and looping through time, a check is made on the emergency_stop variable during the iterations. If the variable is set to one, the Execute button code stops drawing.

The user created function plot_diff is called each minute to plot the change in variance. It takes 2 arguments, the time and the pressure zone selected. The pressure zone is chosen from the current value of the drop-down list. In live operations, the time is the current system time. For research purposes, this is a programmed value. A connection is made to the local SQLEXPRESS database for data to plot. A query is created and executed to return the time, PCA data, and variance that was explained in previous steps outlined in the section entitled Principal Component Operation. These steps are completed in a calc_pca function in the final product included in the appendix. The Execute button code calls this function in addition to plotting the data. The Matlab code inside the calc_pca function includes the following for the query to return this data.

It returns 180 rows of data for the previous 3 hours of calculations. The Matlab code also sets a variable to the amount of rows returned. The amount of data returned indicated by the number of hours in the query can be adjusted as this changes the plot window as well. Three hours were chosen as a start to monitor the variance change. A longer window, such as 24 hours or more would show more of the daily pattern.

```
sqlquery = strcat('select CONVERT(varchar,timestamp,120) as
thetime,currentvalue,variance from [scada].[dbo].[',PZTable,'] where timestamp
<=''',datestr(BeginDateNum,'yyyy-mm-dd HH:MM'),''' and timestamp
>''',datestr(addtodate(BeginDateNum,-180,'minute'),'yyyy-mm-dd HH:MM'),'''');
curs = exec(conn,sqlquery);
curs = fetch(curs);
numrows = size(curs.data,1);
```

Three variables are created and set to data returned from the query. Variable TimeData includes the date/time information, PCAData contains the calculated principal component values, and VarData contains the variance. Another array variable, VarDiff is set to an empty array as well.

A loop from one to the number of rows returned from the query is executed next. This performs the subtraction of the variance data and saves it in variable VarData. At the start of the loop, if the loop variable is set to 1, the first difference calculation is zero since there is nothing to subtract so the first element of the VarDiff array is set to zero. For the remaining items, the array appends the difference between the current variance value and the previous minute data. After the loop is fully executed, it only contains the difference between variance calculations. This is computed with the following Matlab code. Earlier attempts were made to store the difference in the SQLEXPRESS database which would be a waste of storage space. Using a SQL query in the database to calculate the difference was also attempted. This was difficult and time consuming. Instead, it was more efficient to use Matlab's array capabilities to compute the difference.

```
VarDiff = [];
for i = 1:numrows
   if i == 1
      VarDiff = [ 0 ];
   else
```

```
      VarDiff = [ VarDiff ; VarData(i) - VarData(i-1) ];
   end
end
```

Making the actual plot in the plot_diff function is next. Matlab timeseries plots were chosen. An initial timeseries was created with the PCA data and time as inputs. The plot name and time interval of Minute were set for this timeseries. It is then plotted and the color and line width are set as well. Another timeseries is created with the variance difference array, VarDiff, and time as inputs. The name and time interval are set in this timeseries as well. The Matlab command "hold on" is required before plotting the second timeseries. This tells the system to pause while another plot is added to the timeseries. Once added, the color and line width are modified on the new plot. After both timeseries plots have been created, the characteristics of the plot area are defined. The title, x-axis label, y-axis label, and legend are populated. Once the plot is complete, so is the function, which passes back to the calling code to repeat again. The code to plot the data as well as example outputs are below.

```
ts = timeseries(PCAData,TimeData);
ts.Name = 'PCA Data';
ts.TimeInfo.Units = 'Minutes';
pca_plot = plot(ts,'-b');
set(pca_plot,'Color','blue','LineWidth',2);
nts = timeseries(VarDiff,TimeData);
nts.Name = 'Difference Data';
nts.TimeInfo.Units = 'Minutes';
hold on;
diff_plot = plot(nts,'-b');
set(diff_plot,'Color','green','LineWidth',2);
title('PCA Calculations');
xlabel('Time');
ylabel('PCA Value');
legend('PCA','Difference','Location','northwest');
```

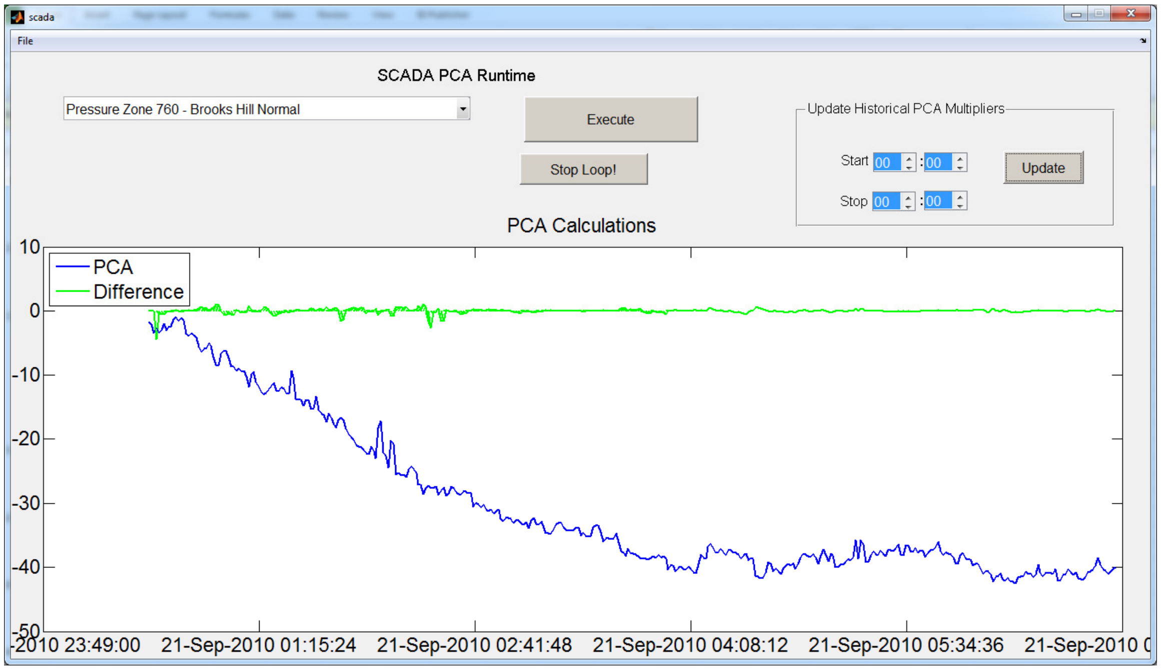


FIGURE 23 - Sample Execution with Normal Operation

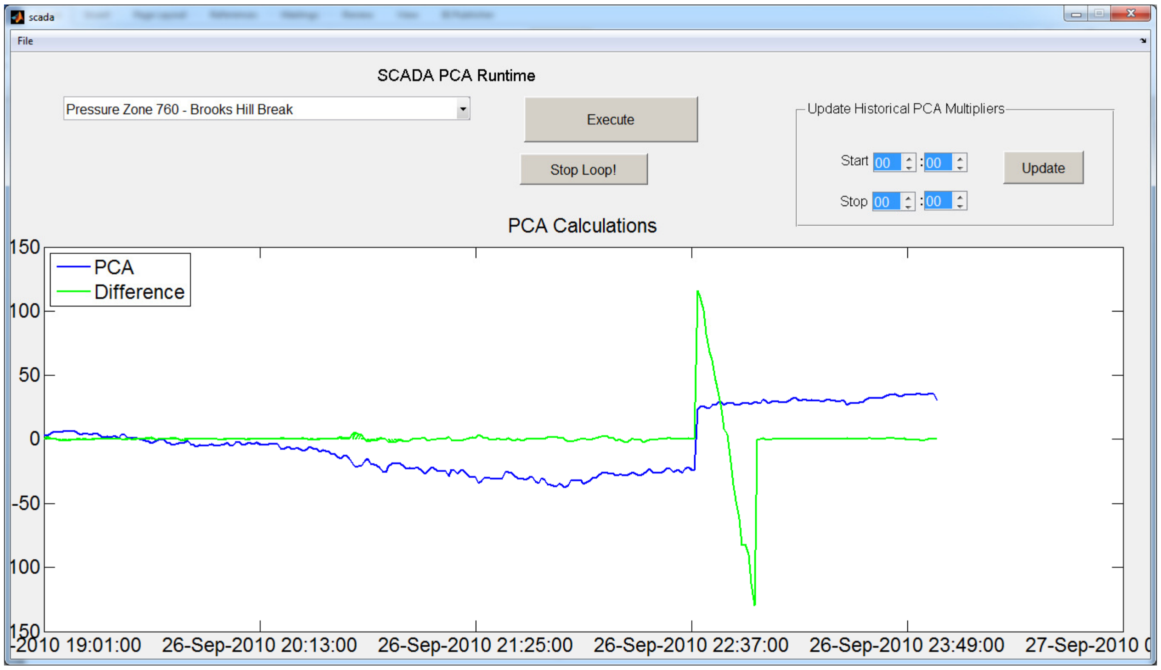


FIGURE 24 - Sample Execution with Break Condition

The runtime GUI illustrates very well how the comparison of the difference in variance can signal an issue in the system. Under good operating conditions, the change in difference is very low and flat. Once an issue occurs, whether from the change in electrical usage or a break, the change in variance is an obvious indicator. When reviewing data for a small area or pressure zone, it is easy to geographically isolate the area of concern. When plotting data for the entire system, it does not provide a geographical area of reference. It will signal that a change is about to occur and provide additional time for close inspection of the raw SCADA information. Since the raw SCADA data also contains typical usage patterns, this can pinpoint a more appropriate geographic area to concentrate efforts.

VI. CONCLUSIONS AND RECOMMENDATIONS

The goal was to implement principal component analysis on a large amount of data with many variables for early detection of data fluctuations. The SCADA system at Louisville Water Company was an excellent match for this research. Each individual variable, or tag, of the raw SCADA data can be plotted which results in a steady pattern over time under normal circumstances. There are hundreds of tags in the system that record different qualities of data. For instance, some tags refer to tank levels stored in feet, some are based on pressure in pounds-per-square-inch, and others measure flow in million-gallons-per-day. With so much data with a wide range of values, it is impossible to plot everything together in one chart to measure operational status. Principal component analysis reduces all applicable variables, or tags in this instance, to a single element. This principal component is representative of all data. It too contains a steady pattern over time under normal conditions. The reduction in variables allows for a single plot to represent the overall status of the system. Monitoring the changes in variation of the principal component can then be used to detect abnormalities in normal operations.

As with any research, numerous attempts with many successes and failures occurred. There were many changes in program design and flow. Typical learning curves with new software such as Matlab and nuances with SQL Server Express caused minor setbacks at times. Certain issues were related to choosing the best solution for a problem. Multiple SQL queries were first executed directly in the database to return numeric information but that proved inefficient compared to the faster processing of a single query result in Matlab. Both the SQL Query and Matlab code provided the same result but Matlab was faster.

This research concludes there are several decisions that affect the output of the principal component calculation. Missing data can occur such as when a pump is taken offline for maintenance. Choosing to replace missing data with an average such as when importing raw SCADA reading or to remove missing data affects the outcome. When calculating the principal coefficients, any missing data was removed. The variance calculated on the principal components was tested with several ranges of data until a 20 minute range seemed to provide the best result. This seemed to be the most significant decision on how well abnormalities could be detected in the data.

The use of principal components to evaluate SCADA data was successful. The amount of change in variance illustrated whether or not something was occurring in the system such as a main break. This was verified by examining raw data for specific incidents and matching it to the principal component data. The principal component data was also able to show the possible early detection of issues in the distribution system. This research used approximately 8 months of data over a 2 year period. During this time, five specific dates were chosen that were either considered a normal operating day or had a major main break. On the normal operating days, the PCA observations did not show any major events outside of the known regular operating indicators. All of the dates with major main breaks did show changes in the PCA variation several days prior to the established event date. However, a knowledgeable workforce is still required to interpret and react on data. Trends do exist in the data but they can alter depending on the time of year and demand. Employees need to know about operational activities such as taking major pumps or sections of main offline for maintenance and what that may impact. The method to use electricity efficiently causes spikes in the data at regular times of the day as well as just the typical power cycling of various distribution components. These changes show in the variance throughout the day even under normal operating conditions such as the large spikes around midnight due to pumping schedules. Staff needs to understand these normal spikes or changes which are not due to a break. Misinterpreting the data could cause staff to investigate system issues when none exist. This can be a great support tool but it also needs to be reinforced with an understanding of the distribution system.

Future enhancements can also be added. A deeper inspection of the tags used in the system as well as how they are grouped might lead to improved detection. Adjusting the graphical user interface to include the overview plot with all SCADA tags and including another plot window showing a user selected pressure zone would provide more location information on where a break or issue was occurring. When running the routine to update PCA coefficients, since it is a very time consuming activity, it should be started in a separate instance of the program. As new SCADA monitoring points are added to the environment, improved detection is possible. The program should also adapt to new SCADA monitoring points without changing source code. Further investigation into the normal operations that cause drastic changes in variance could lead to an improved solution where they do not get highlighted reducing false positives.

APPENDIX I

MATLAB CODE TO INSERT MULTIPLE RECORDS

```
% BeginDate 0:01 to EndDate + 1 0:00
BeginDate='2010-10-05 0:00';
EndDate='2010-10-08 23:59';

CurrentDate = datenum(BeginDate);
StopDate = datenum(EndDate);

conn = database('SQLEXPRESS','','');

while CurrentDate <= StopDate

   TempStart = datestr(CurrentDate,'yyyy-mm-dd HH:MM');
   TE = addtodate(CurrentDate,1,'day');
   TempEnd = datestr(TE,'yyyy-mm-dd HH:MM');
   disp(strcat('Start: ',TempStart,'  Stop: ',TempEnd));

   sqlquery = strcat('insert into [SCADA].[dbo].[ihRawData] (TagName, TimeStamp,
Value, Quality) SELECT TagName, TimeStamp, Value, Quality FROM OPENQUERY
(iHist,''SET
RowCount=0,StartTime=''''',TempStart,''''',EndTime=''''',TempEnd,''''',IntervalMilliseconds=1
Minute,SamplingMode=Calculated,CalculationMode=Maximum SELECT * FROM
ihRawData WHERE TagName = CHFP.BROOKS_STATION_TANK_LEVEL.F_CV or
TagName = CHFP.JEFF_FOREST_TANK_LEVEL.F_CV or TagName =
CHFP.MARTIN_STATION_TANK_LEVEL.F_CV or TagName =
CHFP.JEFF_DISCHARGE_PRESSURE.F_CV)'')');
   disp(sqlquery)

  curs = exec(conn,sqlquery);
  curs = fetch(curs);

   CurrentDate = addtodate(CurrentDate,1,'day');
end

close(conn);
```

APPENDIX II

MATLAB GUI CODE

```
function varargout = scada(varargin)
% SCADA M-file for scada.fig
%      SCADA, by itself, creates a new SCADA or raises the existing
%      singleton*.
%
%      H = SCADA returns the handle to a new SCADA or the handle to
%      the existing singleton*.
%
%      SCADA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in SCADA.M with the given input arguments.
%
%      SCADA('Property','Value',...) creates a new SCADA or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before scada_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to scada_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help scada

% Last Modified by GUIDE v2.5 14-Nov-2011 22:37:24

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @scada_OpeningFcn, ...
                   'gui_OutputFcn',  @scada_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before scada is made visible.
function scada_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to scada (see VARARGIN)

% Choose default command line output for scada
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% This sets up the initial plot - only do when we are invisible
% so window can get raised using scada.
if strcmp(get(hObject,'Visible'),'off')
   plot(1);
end

% UIWAIT makes scada wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = scada_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
global emergency_stop;
emergency_stop = 0;

axes(handles.axes1);
cla;

popup_sel_index = get(handles.popupmenu1, 'Value');
switch popup_sel_index
   case 1 % ALL
      BeginDate='2011-07-06 6:15';
      EndDate='2011-07-06 10:00';
      PZTable='PZALL';
      SourceTable = '[ihist].[dbo].[ih_july]';

      CurrentDate = datenum(BeginDate);
      StopDate = datenum(EndDate);

      while CurrentDate <= StopDate && emergency_stop == 0
         TempCurrentDate = datestr(CurrentDate,'yyyy-mm-dd HH:MM');

         % Calculate PCA and variance for current time
         calc_pca(TempCurrentDate,PZTable,SourceTable);

         % PLOT PCA and variance difference for current time
         plot_diff(TempCurrentDate,PZTable);
         pause(2);

         CurrentDate = addtodate(CurrentDate,1,'minute');
      end
      % plot(sin(1:0.01:25.99));

   case 2 % 660
      msgbox('Not done yet!','Future Enhancement!','help');

   case 3 % 690
      %bar(1:.5:10);
      msgbox('Not done yet!','Future Enhancement!','help');

   case 4 % 760 Brooks Hill Break

      BeginDate='2010-09-26 22:00';
      EndDate='2010-09-26 23:59';
      PZTable='ORIG4';
      SourceTable='[SCADA].[dbo].[ihrawdata]';
```

```
        CurrentDate = datenum(BeginDate);
        StopDate = datenum(EndDate);

        while CurrentDate <= StopDate && emergency_stop == 0
            TempCurrentDate = datestr(CurrentDate,'yyyy-mm-dd HH:MM');

            % Calculate PCA and variance for current time
            calc_pca(TempCurrentDate,PZTable,SourceTable);

            % PLOT PCA and variance difference for current time
            plot_diff(TempCurrentDate,PZTable);
            pause(2);

            CurrentDate = addtodate(CurrentDate,1,'minute');
        end
        %plot(membrane);

    case 5 % 760 Brooks Hill Normal

        BeginDate='2010-09-21 3:30';
        EndDate='2010-09-21 6:59';
        PZTable='ORIG4';
        SourceTable='[SCADA].[dbo].[ihrawdata]';

        CurrentDate = datenum(BeginDate);
        StopDate = datenum(EndDate);

        while CurrentDate <= StopDate && emergency_stop == 0
            TempCurrentDate = datestr(CurrentDate,'yyyy-mm-dd HH:MM');

            % Calculate PCA and variance for current time
            calc_pca(TempCurrentDate,PZTable,SourceTable);

            % PLOT PCA and variance difference for current time
            plot_diff(TempCurrentDate,PZTable);
            pause(2);

            CurrentDate = addtodate(CurrentDate,1,'minute');
        end
        %plot(membrane);

    case 6 % 900
        msgbox('Not done yet!','Future Enhancement!','help');
end

% ---------------------------------------------------------------
```

```
function FileMenu_Callback(hObject, eventdata, handles)
% hObject    handle to FileMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function OpenMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to OpenMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
file = uigetfile('*.fig');
if ~isequal(file, 0)
   open(file);
end

% --------------------------------------------------------------------
function PrintMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to PrintMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
printdlg(handles.figure1)

% --------------------------------------------------------------------
function CloseMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to CloseMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
selection = questdlg(['Close ' get(handles.figure1,'Name') '?'],...
                     ['Close ' get(handles.figure1,'Name') '...'],...
                     'Yes','No','Yes');
if strcmp(selection,'No')
   return;
end

delete(handles.figure1)


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1
% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

set(hObject, 'String', {'ALL', 'Pressure Zone 660', 'Pressure Zone 690', 'Pressure Zone 760
- Brooks Hill Break', 'Pressure Zone 760 - Brooks Hill Normal','Pressure Zone 900'});


% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over pushbutton1.
function pushbutton1_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selection change in start_hour_listbox.
function start_hour_listbox_Callback(hObject, eventdata, handles)
% hObject    handle to start_hour_listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns start_hour_listbox contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from start_hour_listbox
global start_hour_lb;
```

```
start_hour_lb_selected = get(hObject, 'Value');
start_hour_lb_list = get(hObject, 'String');
start_hour_lb = start_hour_lb_list(start_hour_lb_selected);
% msgbox(start_hour_lb);


% --- Executes during object creation, after setting all properties.
function start_hour_listbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to start_hour_listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

set(hObject, 'String',
{'00','01','02','03','04','05','06','07','08','09','10','11','12','13','14','15','16','17','18','19','20','21',
'22','23'});


% --- Executes on button press in monitor_pushbutton.
function monitor_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to monitor_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in upd_hist_pca_pushbutton.
function upd_hist_pca_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to upd_hist_pca_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%update_historical_pca();
cla;
global start_hour_lb;
global start_min_lb;
global stop_hour_lb;
global stop_min_lb;

StartTime=strcat(start_hour_lb,':',start_min_lb);
EndTime=strcat(stop_hour_lb',':',stop_min_lb);
```

```
StartTimeNum = datenum(StartTime);
EndTimeNum = datenum(EndTime);

if EndTimeNum <= StartTimeNum
   msgbox('End Time Can Not Be The Same Or Earlier Than Start Time!');
else
   update_historical_pca();
end;


function update_historical_pca()
global start_hour_lb;
global start_min_lb;
global stop_hour_lb;
global stop_min_lb;

alerts_exist = 0;
alert_fid = fopen('c:\kevin\thesis\gui\alerts.txt', 'a');
log_fid = fopen('c:\kevin\thesis\gui\log.txt', 'a');

fprintf(log_fid, 'Update PCA Multiplier Start Run: %s\n', datestr(now,'yyyy-mm-dd
HH:MM'));

%StartTime='22:00';
%EndTime='23:59';
StartTime=strcat(start_hour_lb,':',start_min_lb);
EndTime=strcat(stop_hour_lb',':',stop_min_lb);
%msgbox(strcat(StartTime,'_to_',EndTime));

StartTimeNum = datenum(StartTime);
EndTimeNum = datenum(EndTime);

fprintf(log_fid, 'Update PCA Multiplier Range:
%s\n',strcat(datestr(StartTimeNum,'HH:MM'),'_to_',datestr(EndTimeNum,'HH:MM')));

% setup waitbar to show progress
wb = waitbar(0,'Calculating PCA ...');
step = 0;
total_reps=abs(EndTimeNum - StartTimeNum)*24*60 + 1;

% Connect to SQL Server Express Database
conn = database('SQLEXPRESS','','');

% Get the tags from the pressure zone
sqlquery = strcat('select tagname from [scada].[dbo].[pressure_zones] where
PressureZone = ''PZALL'' order by tagname;');
```

```
curs = exec(conn,sqlquery);
curs = fetch(curs);
TagList = curs.data;
NumTags = size(curs.data,1);

% create string including all tags to be searched
TagString='(''';
for i = 1:NumTags
   if i == 1
         TagString = strcat(TagString,TagList(i),'''');
   else
      TagString = strcat(TagString,',''',TagList(i),'''');
   end
end
TagString = strcat(TagString,')');
%disp(TagString);

while StartTimeNum < EndTimeNum
   %disp(datestr(StartTimeNum,'HH:MM'));
   %disp(datestr(now,'HH:MM:SS'))

   step = step + 1;
   waitbar(step / total_reps);

   % Get raw data source database and source table
   if (StartTimeNum >= datenum('0:00')) && (StartTimeNum < datenum('3:00'))
      SourceDatabase = 'RAW0';
      SourceTable = 'RAW0';
   elseif (StartTimeNum >= datenum('3:00')) && (StartTimeNum < datenum('6:00'))
      SourceDatabase = 'RAW0';
      SourceTable = 'RAW1';
   elseif (StartTimeNum >= datenum('6:00')) && (StartTimeNum < datenum('9:00'))
      SourceDatabase = 'RAW0';
      SourceTable = 'RAW2';
   elseif (StartTimeNum >= datenum('9:00')) && (StartTimeNum < datenum('12:00'))
      SourceDatabase = 'RAW0';
      SourceTable = 'RAW3';
   elseif (StartTimeNum >= datenum('12:00')) && (StartTimeNum < datenum('15:00'))
      SourceDatabase = 'RAW1';
      SourceTable = 'RAW4';
   elseif (StartTimeNum >= datenum('15:00')) && (StartTimeNum < datenum('18:00'))
      SourceDatabase = 'RAW1';
      SourceTable = 'RAW5';
   elseif (StartTimeNum >= datenum('18:00')) && (StartTimeNum < datenum('21:00'))
      SourceDatabase = 'RAW1';
      SourceTable = 'RAW6';
```

```
  elseif (StartTimeNum >= datenum('21:00')) && (StartTimeNum <= datenum('23:59'))
    SourceDatabase = 'RAW1';
    SourceTable = 'RAW7';
  end;

  % Get the number of rows for each tag at the current time (ie 0:00).
  % Use the most recent number of rows from the minimum count returned of
  % all the tags specified.  It is sorted in ascending order, lowest to
  % highest.  The minimum is used so each tag has the same number of
  % entries to compare or else would get matrix errors.

  %select top 1 count(value) as thecount from raw0
  %group by tagname
  %order by thecount asc;
  %sqlquery = strcat('select * from [RAW1].[dbo].[raw7] where timestamp >= ''2010-
03-10 00:02'' and timestamp <= ''2011-07-03 23:59'' and tagname in ',TagString,' and
convert(time,timestamp) = ''',datestr(StartTimeNum,'HH:MM'),''' order by timestamp
asc');
  sqlquery = strcat('select * from [',SourceDatabase,'].[dbo].[',SourceTable,'] where
timestamp >= ''2010-03-10 00:02'' and timestamp <= ''2011-07-03 23:59'' and tagname in
',TagString,' and convert(time,timestamp) = ''',datestr(StartTimeNum,'HH:MM'),''' order
by timestamp asc');
  %disp(cell2mat(sqlquery));
  curs = exec(conn,sqlquery);
  curs = fetch(curs);
  bigdata = curs.data;
  %disp(size(bigdata));

  minrows = 32000;
  for i = 1:NumTags
    cursize = size(find(ismember(bigdata,TagList(i))),1);
    if cursize < minrows
      minrows = cursize;
    end
  end
  %disp(minrows);

  % Initialize empty matrix H for historical data
  H = [];

  % Get data for specific time, ie 0:05
  for i = 1:NumTags

    [r,c] = find(ismember(bigdata,TagList(i)));
    smalldata = sortrows(bigdata(r,:),2);
    singletag = str2double(smalldata(end-(minrows-1):end,3));
```

```
   H = horzcat(H,singletag);
end

% find 0 values and replace with avg from non-zero elements
[r,c]=find(ismember(H,0));
TempH = H;
[Tr,Tc]=size(TempH);
averages=[];
for i = 1:Tc
   V = TempH(:,i);
   [Vr,Vc]=find(ismember(V,0));
   Vrows=sort(Vr,'descend');
   for j = 1:size(Vrows)
      V(Vrows(j),:) = [];
   end
   averages(i)=mean(V);
end

% replace original 0 data with averages
for i = 1:size(r)
   H(r(i),c(i)) = averages(c(i));
end
% end find 0 values and replace with avg from non-zero elements

% n = number of rows
% m = number of columns (ie variables)
[histn histm] = size(H);

% Calculate mean for each column
HMean = mean(H);

% Calculate Standard Deviation for each column
HStd = std(H);

% Calculate normalized data
Hn = (H - repmat(HMean,[histn 1])) ./ repmat(HStd,[histn 1]);

% Check for NaN in matrix due to non-changing variables
% Set HistoricalPCA value to 0 for future processing to not use it and
% remove the tag from current processing
tTagList = TagList;
tNumTags = NumTags;
j = NumTags;
while j >= 1
   if isnan(Hn(:,j))
```

```
            fprintf(alert_fid,'%s\n',(strcat(datestr(now,'yyyy-mm-dd HH:MM'),': Tag
#',num2str(j),' [',cell2mat(tTagList(j)),'] is NaN')));
            alerts_exist = 1;
            %update value with 0 in HistoricalPCA table
            sqlquery=strcat('update [scada].[dbo].[HistoricalPCA] set [',tTagList(j),'] = 0
where Time = ''',datestr(StartTimeNum,'HH:MM'),''';');
            curs = exec(conn,sqlquery);

            %remove Hn and TagList and subtract 1 from NumTags
            tTagList(j)=[];
            Hn(:,j)=[];
            tNumTags = tNumTags - 1;
        end
        j = j - 1;
    end

    % Perform PCA on normalized data

    [HCOEFF HSCORE HLATENT] = princomp(Hn);

    % store HCOEFF into Historical table
    for i = 1:tNumTags
        sqlquery=strcat('update [scada].[dbo].[HistoricalPCA] set [',tTagList(i),'] =
',num2str(HCOEFF(i,1),16),' where Time = ''',datestr(StartTimeNum,'HH:MM'),''';');
        curs = exec(conn,sqlquery);
        %disp(sqlquery);
    end

    % perform checks on HLATENT and manually calculate variance on HCOEFF
    str1 = sprintf('%.4f',sum(HLATENT));
    str2 = sprintf('%.4f',histm);
    str3 = sprintf('%.4f',sum(var(HSCORE)));
    if str1 ~= str2
        fprintf(alert_fid,'%s\n',(strcat(datestr(now,'yyyy-mm-dd
HH:MM'),':',datestr(StartTimeNum,'HH:MM'),': HLATENT sum != number of
columns!')));
        alerts_exist = 1;
        %disp('HLATENT sum != number of columns!');
    end

    if str3 ~= str2
        fprintf(alert_fid,'%s\n',(strcat(datestr(now,'yyyy-mm-dd
HH:MM'),':',datestr(StartTimeNum,'HH:MM'),': HSCORE variance sum != number of
columns!')));
        alerts_exist = 1;
        %disp('HSCORE variance sum != number of columns!');
    end

    % Increase time for loop
    StartTimeNum = addtodate(StartTimeNum,1,'minute');
end
close(wb);

if alerts_exist == 1
    h = msgbox('Alerts generated! Please see alert.log','ALERT','warn');
end

fprintf(log_fid, 'Update PCA Multiplier Stop Run: %s\n', datestr(now,'yyyy-mm-dd
HH:MM'));
fclose(alert_fid);
fclose(log_fid);
close(conn);



% --- Executes on selection change in start_min_listbox.
function start_min_listbox_Callback(hObject, eventdata, handles)
% hObject    handle to start_min_listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns start_min_listbox contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from start_min_listbox
global start_min_lb;
start_min_lb_selected = get(hObject, 'Value');
start_min_lb_list = get(hObject, 'String');
start_min_lb = start_min_lb_list(start_min_lb_selected);


% --- Executes during object creation, after setting all properties.
function start_min_listbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to start_min_listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
set(hObject, 'String',
{'00','01','02','03','04','05','06','07','08','09','10','11','12','13','14','15','16','17','18','19','20','21',
'22','23','24','25','26','27','28','29','30','31','32','33','34','35','36','37','38','39','40','41','42','43','
44','45','46','47','48','49','50','51','52','53','54','55','56','57','58','59'});


% --- Executes on selection change in stop_hour_listbox.
function stop_hour_listbox_Callback(hObject, eventdata, handles)
% hObject    handle to stop_hour_listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns stop_hour_listbox contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from stop_hour_listbox
global stop_hour_lb;
stop_hour_lb_selected = get(hObject, 'Value');
stop_hour_lb_list = get(hObject, 'String');
stop_hour_lb = stop_hour_lb_list(stop_hour_lb_selected);


% --- Executes during object creation, after setting all properties.
function stop_hour_listbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to stop_hour_listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

set(hObject, 'String',
{'00','01','02','03','04','05','06','07','08','09','10','11','12','13','14','15','16','17','18','19','20','21',
'22','23'});


% --- Executes on selection change in stop_min_listbox.
function stop_min_listbox_Callback(hObject, eventdata, handles)
% hObject    handle to stop_min_listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = cellstr(get(hObject,'String')) returns stop_min_listbox contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from stop_min_listbox

global stop_min_lb;
stop_min_lb_selected = get(hObject, 'Value');
stop_min_lb_list = get(hObject, 'String');
stop_min_lb = stop_min_lb_list(stop_min_lb_selected);

% --- Executes during object creation, after setting all properties.
function stop_min_listbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to stop_min_listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

set(hObject, 'String',
{'00','01','02','03','04','05','06','07','08','09','10','11','12','13','14','15','16','17','18','19','20','21',
'22','23','24','25','26','27','28','29','30','31','32','33','34','35','36','37','38','39','40','41','42','43','
44','45','46','47','48','49','50','51','52','53','54','55','56','57','58','59'});

function plot_diff(BeginDate, PZTable)
    BeginDateNum = datenum(BeginDate);

    % Connect to SQL Server Express instance
    conn = database('SQLEXPRESS','','');

    % Get the tags from the pressure zone
    sqlquery = strcat('select CONVERT(varchar,timestamp,120) as
thetime,currentvalue,variance from [scada].[dbo].[',PZTable,'] where timestamp
<=''',datestr(BeginDateNum,'yyyy-mm-dd HH:MM'),''' and timestamp
>''',datestr(addtodate(BeginDateNum,-60,'minute'),'yyyy-mm-dd HH:MM'),'''');
    curs = exec(conn,sqlquery);
    curs = fetch(curs);
    numrows = size(curs.data,1);

    TimeData = curs.data(:,1);
    PCAData = cell2mat(curs.data(:,2));
    VarData = cell2mat(curs.data(:,3));
    % Calculate the difference
    VarDiff = [];
    for i = 1:numrows
       if i == 1
          VarDiff = [ 0 ];
       else
          VarDiff = [ VarDiff ; VarData(i) - VarData(i-1) ];
       end
    end


    % Create plots
    ts = timeseries(PCAData,TimeData);
    ts.Name = 'PCA Data';
    ts.TimeInfo.Units = 'Minutes';
    pca_plot = plot(ts,'-b');
    set(pca_plot,'Color','blue','LineWidth',2);
    nts = timeseries(VarDiff,TimeData);
    nts.Name = 'Difference Data';
    nts.TimeInfo.Units = 'Minutes';
    hold on;
    diff_plot = plot(nts,'-b');
    set(diff_plot,'Color','green','LineWidth',2);

    title('PCA Calculations');
    xlabel('Time');
    ylabel('PCA Value');
    legend('PCA','Difference','Location','northwest');

    % Close database connection
    close(conn);


% --- Executes on button press in stop_button.
function stop_button_Callback(hObject, eventdata, handles)
% hObject    handle to stop_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global emergency_stop;
emergency_stop=1;


function calc_pca(BeginDate, PZTable, SourceTable)
    BeginDateNum = datenum(BeginDate);

    % Connect to SQL Server Express instance
```

```
  conn = database('SQLEXPRESS',",");

  % Get the tags from the pressure zone
  sqlquery = strcat('select tagname from [scada].[dbo].[pressure_zones] where
PressureZone = "',PZTable,"' order by tagname;');
  curs = exec(conn,sqlquery);
  curs = fetch(curs);
  TagList = curs.data;
  NumTags = size(curs.data,1);

  % create string including all tags to be searched
  TagString='("';
  TagHistString='[';
  for i = 1:NumTags
    if i == 1
      TagString = strcat(TagString,TagList(i),"");
      TagHistString = strcat(TagHistString,TagList(i),']');
    else
      TagString = strcat(TagString,',"',TagList(i),"");
      TagHistString = strcat(TagHistString,',[',TagList(i),']');
    end
  end
  TagString = strcat(TagString,')');
  %disp(TagString);

  % Input Historical PCA data into matrix for quicker manipulation
  % The correct row can be calculated by (60 x hour) + (minute) + 1
  % eg 1:33 = (60 x 1) + 33 + 1 = 94 so use row 94 in matrix for that value
  sqlquery = strcat('select CONVERT(varchar,time,108) as thetime,',TagHistString,'
from [scada].[dbo].[HistoricalPCA] order by thetime;');
  curs = exec(conn,sqlquery);
  curs = fetch(curs);
  HISTPCA = curs.data;
  %disp(HISTPCA);

  % Get the current data value for each tag
  % Get the previous hour data value for each tag
  % Get the historical data for the tag at the current time
  % Multiply the current value * the historical PCA value
  % Add it to the total
  CurrentTotal = 0;
  %%%PreviousTotal = 0;

  % Current
```

```
   sqlquery = strcat({'select * from '},SourceTable,{' where tagname in '},TagString,{'
and timestamp ='''},datestr(BeginDateNum,'yyyy-mm-dd HH:MM'),{''' order by
tagname;'});
   % disp(sqlquery);
   curs = exec(conn,sqlquery);
   curs = fetch(curs);
   curdata = curs.data;
   %disp(curdata);

   curHH = str2num(datestr(BeginDateNum,'HH'));
   curMM = str2num(datestr(BeginDateNum,'MM'));
   %disp(strcat(num2str(curHH),':',num2str(curMM)));
   rownum = (60 * curHH) + curMM + 1;
   histrow = cell2mat(HISTPCA(rownum,2:(NumTags + 1)));
   %disp(histrow);

   CurrentTotal = histrow * str2double(curdata(:,3));
   %disp(CurrentTotal);


   % calculate variance for current and prior 20 min
   %ORIG: sqlquery = strcat('select currentvalue-historicalvalue as DIFF from
[scada].[dbo].[PZ1] where timestamp <=''',datestr(BeginDateNum,'yyyy-mm-dd
HH:MM'),''' and timestamp >''',datestr(addtodate(BeginDateNum,-20,'minute'),'yyyy-mm-
dd HH:MM'),''' order by TimeStamp;');
   %%sqlquery = strcat('select currentvalue-historicalvalue as DIFF from
[scada].[dbo].[',PZTable,'] where timestamp <=''',datestr(BeginDateNum,'yyyy-mm-dd
HH:MM'),''' and timestamp >''',datestr(addtodate(BeginDateNum,-20,'minute'),'yyyy-mm-
dd HH:MM'),''' order by TimeStamp;');
   %%%%sqlquery = strcat('select abs(currentvalue-historicalvalue) as DIFF from
[scada].[dbo].[',PZTable,'] where timestamp <=''',datestr(BeginDateNum,'yyyy-mm-dd
HH:MM'),''' and timestamp >''',datestr(addtodate(BeginDateNum,-20,'minute'),'yyyy-mm-
dd HH:MM'),''' order by TimeStamp;');
   sqlquery = strcat('select currentvalue from [scada].[dbo].[',PZTable,'] where timestamp
<=''',datestr(BeginDateNum,'yyyy-mm-dd HH:MM'),''' and timestamp
>''',datestr(addtodate(BeginDateNum,-20,'minute'),'yyyy-mm-dd HH:MM'),''' order by
TimeStamp;');
   curs = exec(conn,sqlquery);
   curs = fetch(curs);
   TheCV = cell2mat(curs.data);
   TheCV = [ TheCV ; CurrentTotal ]; % Add current value to vector since not in table
yet
   TheVar = var(TheCV);
   %disp(TheDiff);
   %disp(TheVar);
```

```
    % insert data into table
    sqlquery = strcat('insert into
[scada].[dbo].[',PZTable,'](TimeStamp,CurrentValue,Variance) values
(''',datestr(BeginDateNum,'yyyy-mm-dd
HH:MM'),''',',num2str(CurrentTotal),',',num2str(TheVar),');');
    %%sqlquery = strcat('insert into [scada].[dbo].[',PZTable,'] (TimeStamp,CurrentValue)
values (''',datestr(BeginDateNum,'yyyy-mm-dd HH:MM'),''',',num2str(CurrentTotal),');');
    %disp(sqlquery);
    curs = exec(conn,sqlquery);

    close(conn);
```

LIST OF REFERENCES

[1] Howard, Greta. 2011. Instructions and Specifications for Request for Proposal for PeopleSoft Managed Services. Louisville Water Company.

[2] James Bates, "Spatial Pipeline Infrastructure Network." *Louisville Water Company GIS Intranet Server*, 29 June 2010, available from http://mars/lwc; accessed 29 June 2010.

[3] "Principal Component Analysis," *Wikipedia*, 23 May 2011, available from http://en.wikipedia.org/wiki/Principal_component_analysis; accessed 23 May 2011.

[4] Bryant, Larry. 2011. Private communication, Louisville Water Company.

[5] GE Proficy Historian. GE Intelligent Platforms. Charlottesville, Virginia.

[6] "Proficy Historian," *GE Intelligent Platforms*, 7 June 2011, available from http://www.ge-ip.com/products/2420; accessed 7 June 2011.

[7] Jolliffe, I.T., 2002. *Principal Component Analysis*. Springer.

[8] "SCADA," *Free Dictionary*, 14 May 2004, available from http://encyclopedia.thefreedictionary.com/SCADA; accessed 14 May 2004.

[9] Dearing Smith, Kelley, 2010. *Water Works, 150 Years of Louisville Water Company*. Butler Books.

[10] Smith, Lindsay I. 2002. A Tutorial on Principal Components Analysis.

[11] "Normalization (statistics)," *Wikipedia*, 26 July 2011, available from http://en.wikipedia.org/wiki/Normalization (statistics); accessed 26 July 2011.

[12] "Eigenvector," *Dictionary.com*, 25 July 2011, available from http://dictionary.reference.com/browse/eigenvector; accessed 25 July 2011.

[13] "Eigenvalues and eigenvectors," *Wikipedia*, 27 July 2011, available from http://en.wikipedia.org/wiki/Eigenvalues and eigenvectors; accessed 27 July 2011.

VITA

Kevin Kastensmidt was born and raised in Louisville, KY. He enrolled at the University of Louisville in the Fall 1991 semester. Before graduating with his bachelor degree, he married. In Spring 1995 he completed his Bachelor of Science in Speed School of Engineering. Following graduation, he began full-time employment at Louisville Water Company starting as a Systems Analyst. While at Louisville Water Company, Kevin has managed numerous systems including Windows, Unix, Linux, Oracle, and SQL Server among others. Today, he is Manager of Technical Services.

While working full time, Kevin continued his education at Speed School to pursue his Master of Engineering Degree only able to take a class or two each year if that. During this time, he and his wife Amy had 3 children which keep them very busy!

After much persistence and support from Amy and his children, he has been able to obtain his Master of Engineering Degree!