The Society for engineering in agricultural, food, and biological systems

This is not a peer-reviewed article.

Paper Number: 033089
An ASAE Meeting Presentation

An Object-Oriented Architecture for Field Data Acquisition, Processing and Information Extraction

Dev S. Shrestha, Brian L. Steward, Cory Van Wyngarden

Iowa State University, Agricultural and Biosystems Engineering Department

139 Davidson, Ames, IA-50011, USA

Written for presentation at the
2003 ASAE Annual International Meeting
Sponsored by ASAE
Riviera Hotel and Convention Center
Las Vegas, Nevada, USA
27- 30 July 2003

Abstract. *Software architecture was developed to automate site specific field data acquisition, processing, and geo-referenced crop plant parameters extraction. The architecture supported acquisition and processing of different data streams such as digital video for machine vision and digital serial communications of NMEA strings. The number of channels for data import could be easily expanded for multiple video, GPS, and other signal sources. The architecture was object-oriented and each component in the architecture was developed as a separate class. A key component of this architecture was a supervisor class, which communicated and coordinated the operations on all other classes. Based on this framework, early stage corn population estimation (ESCOPE) software was developed which grabs pre-recorded digital video from a vehicle-mounted camera, that was passed over corn rows, and acquires GPS strings which were modulated and recorded on the audio channel. A digital video (DV) capture class was written to acquire video using Microsoft's DirectShow® technology which enables camera control and video acquisition using higher level hardware functions. After completion of software development, reusability and extensibility characteristics were demonstrated by adding a class to acquire images from the hard drive and also by deriving a new image analyzer class to extract an additional feature. The architecture forms a general framework for developing reusable and extensible software for field data sensing systems.*

Keywords. Precision agriculture, machine vision, object-oriented software architecture, image processing, unified modeling language, field sensing

Introduction

Precision agriculture (PA) is an important technological development in contemporary agriculture for managing the spatial variability that naturally occurs in crop production (Schueller et al., 2002). The National Research Council (1997) refers to PA as a management strategy that uses information technologies to bring data from multiple sources to bear on decisions associated with crop production. The key idea behind PA is to measure and manage in-field variability to optimize the crop production system. Li et al. (1998) categorized variability into six categories – yield variability, field variability, soil variability, crop variability, variability in anomalous factors, and management variability.

Past research has developed models to predict the effect of different variables on crop yield. However, most crop models account for a small number of factors that may limit yields in the field (Gary et al., 1998; Irmak et al., 2001; Schueller et al., 2002). Pierce and Nowak (1999) point out that currently no complete precision agriculture system exists; rather various components of traditional crop management systems have been addressed separately regarding their potential for site-specific management. To face the challenge of diversity, modelers will certainly have to adopt more generic approaches (Gary et al., 1998). Part of the reason that researchers were bound to include only a small set of variables in their model was that the characterization of field-level variability had been generally inaccessible or prohibitively expensive to acquire. Advances in electronics, communications, and software over the past several decades have removed those earlier impediments. Inexpensive sensors and microprocessors coupled with integrating software now enable agricultural producers to collect vast amounts of geo-referenced data (Schmoldt, 2001).

Different sensors have been developed to measure field scale variability in several crop production parameters. Yield sensors for major crops are approaching maturity and are commercially available (Zhang et al., 2002). Sensors have been developed for measuring soil properties such as soil organic matter and moisture content (Hummel et al., 2001), electrical conductivity (Lund et al., 2000) and nutrients (Birrell and Hummel, 2000). Sensors have also been developed to measure plant parameters such as corn population (Birrell and Sudduth, 1995), N-status (Goel et al., 2003) and leaf area index (Johnson et al., 2003). While not applied to field scale deployment, image-based crop growth measurement has been shown to be effective in measuring and modeling crop plant growth in laboratory or greenhouse applications (Morden et al., 1997; Tarbell et al., 1991; Tarbell and Reid, 1991; Van Henten and Bontsema, 1995). Machine vision based algorithms have been developed to estimate several plant growth parameters such as plant shape and size, plant spacing, plant height and leaf color (El-Faki et al., 2000; Nishiwaki et al., 2001; Shrestha and Steward, 2003; Tang et al., 2000).

Bottlenecks in successful application of PA include a lack of (1) developed sensing technologies needed to adequately characterize field-scale spatial variability, (2) a flexible data acquisition and processing system that can be deployed in a field to gather and process data, and (3) agronomic knowledge relating crop inputs and plant response to those inputs. This work focuses on the second of these bottlenecks.

Considering these facts, it is important to develop a system which can acquire and combine different types of geo-referenced data simultaneously and process these data to extract plant growth and soil parameters. An ideal system would receive data from different information channels, then process the information as it arrives, and extract the parameters of interest. The system should also be able to accommodate future developments with minimal effort and be reusable when new algorithms are added to the software.

Developing a software system architecture prior to its construction or renovation is as essential as having a blueprint for a large building. There are many advantages to developing system

architecture. First, it breaks down the entire task into individual standalone modules so that an individual or small group of individuals can work on a component. Second, it offers ease in management of individual components. Third, architecture facilitates reuse, modification and improvement of the software. Fourth, the architecture helps isolate errors and allows individual module testing. Fifth, it improves readability and brings clarity.

A growing body of literature illustrates the utility of object-oriented architectures in agricultural applications. Object-oriented architectures are being investigated in the area of crop growth modeling (Beck et al., 2003; Pan et al., 2000). The development of agricultural autonomous vehicles is being facilitated through the use of object-oriented architectures which is helping to maximize the development of dispersed research groups (Blackmore et al., 2002; Sorensen et al., 2002; Torrie et al., 2002; Will et al., 2002). These architectures enable reuse of software components and facilitate communication between collaborating research groups. In addition, object-oriented design enables design abstraction and systematic thinking about highly complex systems.

The objective of this research was to develop a software architecture for a data acquisition and processing system which can receive data from many sources, process these data, and output desired parameter estimates. Based on this architecture, an early stage corn population estimation (ESCOPE) system was developed. ESCOPE grabs pre-recorded digital video from a vehicle-mounted camera that was passed over corn rows and acquires GPS strings which were modulated and recorded on the audio channel. Using ESCOPE as a case study, we will show how this architecture has the following characteristics:

1. Expansibility of data acquisition from different input data channels through the use of standard hardware interfaces.
2. Reusability of the developed algorithm.
3. Extensibility through ease in incorporating new data processing algorithms.

This architecture would be particularly beneficial for several different groups of people. The first group of the people is researchers trying to extract a crop growth parameter from video such as crop greenness, spectral reflectance, plant spacing, plant height, or top projected canopy area. The benefit for this group of people comes from reducing the distractions typically encountered when introducing new information streams or processing algorithms into a system. The architecture helps to develop such a system with minimal overload as it is designed to use a standard hardware interface which greatly reduces the unnecessary distractions from a research point of view. Another group is crop modelers who need different field level variability data to calibrate models, develop input prescriptions, and assess economic and environmental risks. As PA is further developed, crop consultants and agricultural producers will benefit from such a system as a component within a larger decision support system.

System Architecture

An object-oriented programming (OOP) approach was followed because of its many advantages over structural programming. In general, OOP uses software objects that resemble real world objects which are easier to understand and conceptualize. OOP also enables the reuse of an object and easy modification through class inheritance and is less prone to error because of its capability of data encapsulation. Object-oriented systems can be effectively described using Unified Modeling Language (UML). UML is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems (Object Management Group, 2003). UML is widely accepted by the software engineering community for constructing and analyzing the early stages of system

development (Aleman and Alvarez, 2000). The architecture is described in three different steps as system activities, use case scenario, and class diagram.

System Activities

The chronological activity of the entire system is shown in a flow chart-like diagram called the context diagram. For a field data acquisition, processing and information extraction system, the system must first read the field data recorded in the form of magnetic tape or other digital formats (fig. 1). In order for the system to be flexible, it should be able to accept different types of data from different ports. The data could be one-dimensional numeric data like soil temperature or two-dimensional data like an image, or it could be a string of characters like the NMEA string associated with the GPS signal. For all three different scenarios, the system being developed should accept, parse, and correlate data from different channels.

Once the data are read from the port, it is temporarily stored in computer RAM. Processing of stored data can be done in two different modes, batch mode and continuous mode. In the batch mode, a chunk of data is stored in memory and then processed. The external devices have to be paused during data processing. Only after completion of processing, the computer RAM is freed and another chunk of data is logged. This architecture is simple in that processing of one chunk of data is completed before another process starts, and sequence of processing is well defined. If processing time is longer than capturing time, then the user must do batch processing of data. In the continuous mode, the processing and capturing of data occurs in parallel. The data is read to fill a portion of computer memory, and the system starts processing that data; while processing, the computer reads in more data into another part of the memory. As soon as data processing for the first part of memory is finished, it starts processing the next chunk of data. This mode is suitable for real time processing and is feasible only when the processing time is less than or equal to the data capture time.

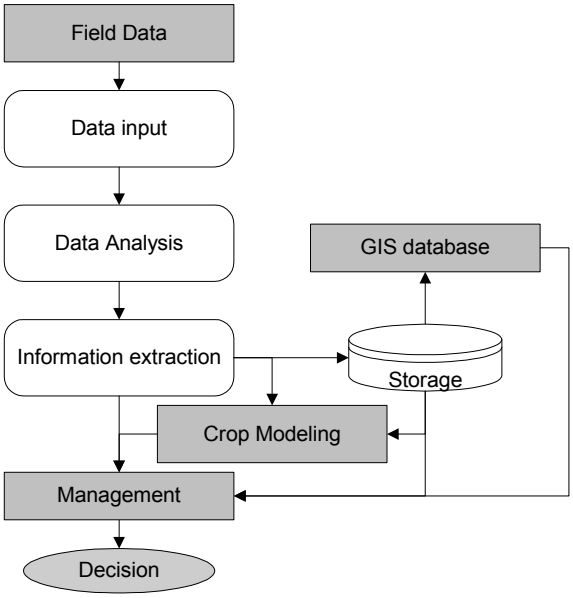

Figure 1. Context diagram for field data acquisition, processing and information extraction system. The software architecture developed acquires recorded data, analyzes, produces site-specific information, and stores into hard drive (Non-grayed boxes).

In either case, once the analysis of the captured data is completed, the system extracts the site-specific information and then passes it to the storage system; usually a hard drive (fig. 1). The extracted information may be used in a crop model to establish the relationship between yields

and a set of indicator variables. This relationship may be used by farmers to optimize crop production in the field. The development of a reliable crop model may require several years of data; crop management, however, may also use single year data combined with prior experiences for management decisions. The information may also be used to produce a GIS database, which can be used by farm management for field variability mapping.

Use-case scenario

A use-case model attempts to graphically depict the system users – called actors – and actions – called use-cases. The main actors of the system are digital data, video and GPS signals, researchers and management. In a typical field application scenario, the system reads different data (fig. 2); researchers and farm managers then decide on what kind of information is extracted after processing. The choices of type of information to extract depend on the type of field data and the capability of the software. Researchers may also add new processing modules into the system to extract more information. The management can make a future field-management decision based on the extracted information.

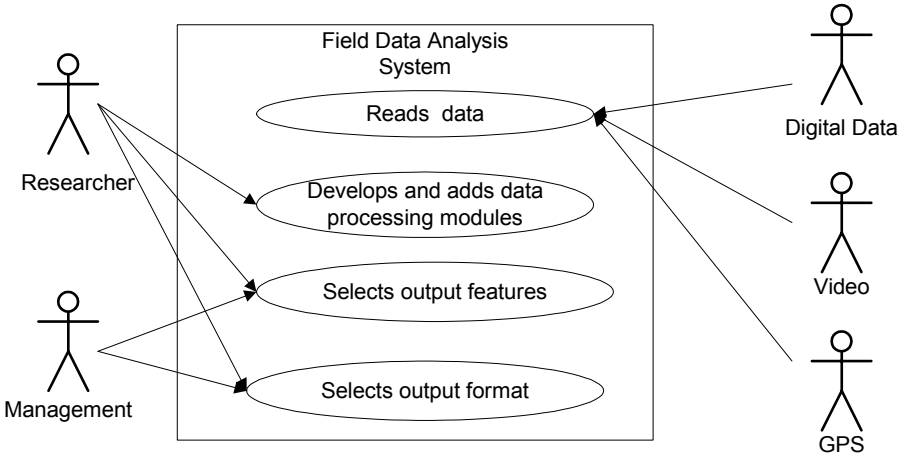

Figure 2. Use-case diagram. The field data analysis system interacts primarily with these actors: digital data, Video, GPS, researcher and management.

Class Diagram

Class diagrams are used to show the relationship among systems objects. A group of activities pertaining to some specific task was identified from a use-case scenario and developed as a separate class. In order to synchronize and to instruct each of these classes to perform in harmony, a Supervisor class was designed. When the program was initiated, the Supervisor class was created, and this class, in turn, initialized all other classes as needed (fig. 3). The components of the system architecture are described in the following sections.

Grabber Class

Standardization of hardware and software communication protocols has made program code reusable. One piece of software can use the service provided by some other software without having to understand the details of that software. The Component Object Model (COM) technology developed by Microsoft® (Redmond, WA) is a software model which provides a standard protocol for object intercommunication and reusability (Root and Boer, 1999). It is much easier and safer to use standard components than writing a new program to do the same

task. The DVCapture class takes advantage of freely available DirectShow® technology which is a COM object for FireWire communication.

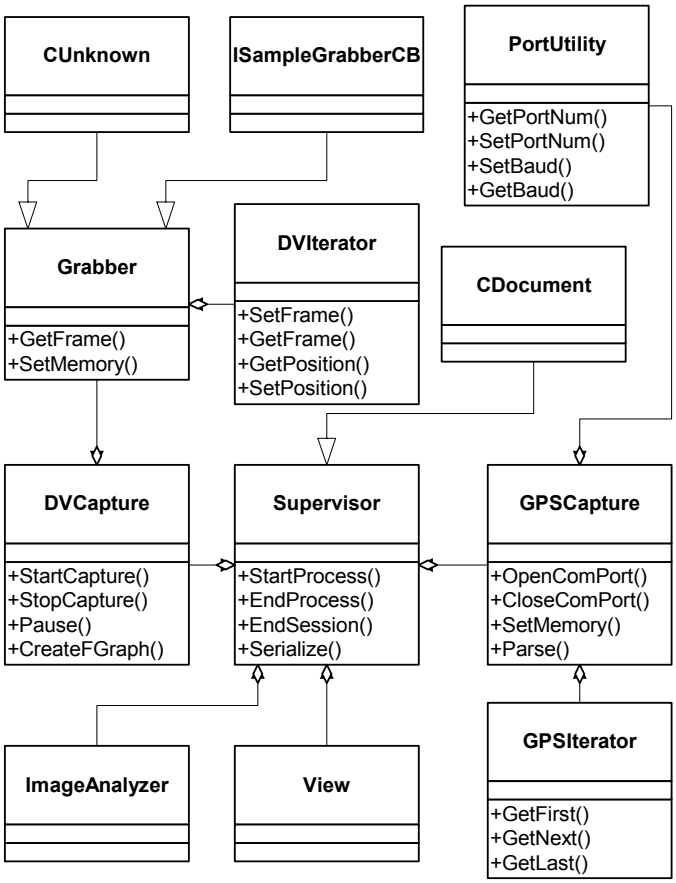

Figure 3. Class diagram. Supervisor class is the heart of the structure, which manages all other classes and activities.

COM is a platform-independent object-oriented system for creating software components that can interact. COM objects can be created with a variety of programming languages. Each COM object is an instance of a particular class, and each supports one or more interfaces. Each interface provides a set of functionality that the COM object provides. An interface specifies the interface's member functions, calling methods, their return types, the number and types of their parameters, and what they must do (MSDN Documentation, 2003c).

DirectX® (Microsoft Corp., Redmond, WA) consists of a group of COM objects, which provides a standard development platform for Windows-based PCs by enabling software developers to access specialized hardware features without having to write hardware-specific code. This enables developers to spend more time on working on the features of the application, and less time worrying about the different hardware configurations. The hardware capabilities of each device can be queried at the application run time, giving the application the ability to adjust itself to specific hardware implementations.

DirectShow is one of the member components of DirectX. DirectShow technology can be used to control video cameras for playback and image acquisition. DirectShow automatically detects and uses video and audio acceleration hardware when available, but also supports systems

without acceleration hardware. DirectShow simplifies media playback, format conversion, and capture tasks.

DirectShow divides the processing of multimedia tasks such as video playback into a set of steps known as filters. Filters can be used in sequence to perform a specific task. A filter graph is developed to capture video and to split it into the individual frames (MSDN Documentation, 2003b).

The Grabber class is derived from the DirectX base class, CUnknown, and the ISampleGrabberCB interface. This class was designed for communicating with a digital camcorder through FireWire. Since this class is derived from CUnknown, it is a COM object by itself. This strategy allows some call back methods of ISampleGrabberCB to be overwritten for user-defined memory management. It also allows the absolute track number of the video frame being acquired to be assigned so frame drops can be detected during image acquisition.

DVCapture Class

To call the camera controlling function, it is necessary that the COM object first be created. The DVCapture class creates the Grabber COM object. A filter graph is created in this class to capture video frames. It is helpful to think of a filter graph as a specialized class for video parsing. Once the filter was built, different functions were written so that more intuitive camera commands like StartCapture and StopCapture could be used to control the camera from the Supervisor class. Some of the important tasks this class performs are 1) finding video input devices, 2) watching for events such as end of tape or stop camera, 3) getting video streams and splitting them into frames, 4) playing, pausing, and stopping the camera, and seeking to a location.

DVIterator Class

The DVIterator class serves as a container class for captured image frames. On initialization, the Grabber class creates this class and passes a pointer to this class to the DVCapture class. In turn, the DVCapture class passes back the pointer to the Supervisor class, so that the Supervisor class can pass it to any class that requests the image storage location. This class keeps track of the image frames in computer RAM, so that the frames can be retrieved in sequence for processing. In addition, once the processing is completed, memory can be freed for storing new data. Some of the tasks done by this class are 1) moving to the first or last image frame, 2) reading the frame from a specified location, and 3) replacing a frame at a specified location.

PortUtility class

The PortUtility class initializes serial port settings. In this architecture, the software is expected to read the digital data like GPS and other measurements through serial ports. This class relieves programmers from writing their own serial port setting class repeatedly for each channel.

GPSCapture class

The GPSCapture class uses the PortUtility class to capture the digital data from the serial port. The GPSCapture class then initializes the GPSIterator class to store the data. This class also provides the functionalities to parse NMEA strings.

GPSIterator Class

The GPSIterator class serves as a container to store GPS data. This class is created by the GPSCapture Class, which holds a pointer to this class and passes this pointer back to the Supervisor class so that the Supervisor class can pass it to the classes that act on GPS data. The GPSIterator class provides functionalities to step through the acquired GPS data in RAM. The function of this class is similar to the DVIterator class except that the class handles GPS data rather than video data.

Supervisor Class

The Supervisor class is the main controller of all classes. It initiates, controls and releases all other classes as needed. This class holds a pointer to each of the classes it creates and keeps track of all the events of the application. The Supervisor class was derived from the Microsoft Foundation Class (MFC) CDocument class (MSDN Documentation, 2003a). This inherits many functionalities of the CDocument class to manage the application data and share the data with the View class.

All of the Windows messages are handled through the Supervisor class. The Windows messages are routed to one of the relevant class functions and then depending on that message type and class's response, the Supervisor takes the next action. When the user runs the software to acquire data from selected sources, the Supervisor class creates relevant capture classes which checks the hardware for its readiness. If the hardware is not ready, the Supervisor class displays a warning message. The user can turn on or off any of the data input channels. The Supervisor class communicates with other classes through messaging system. The action taken by Supervisor class strictly depends on the type of message it receives. For instance, while receiving the images from a camcorder, the DVCapture class may send the end of tape message to the Supervisor class. Then the Supervisor class calls the function pertaining to that specific message. Each step from reading the data to final output of information is coordinated by this class. The message handling functions are made virtual so that if a new class is added in the architecture which needs a different message handling method, a new Supervisor class can be derived from the old Supervisor class to overwrite the message handling functions.

Depending on the data channels selected, the Supervisor class allows the user to choose from the list of information that the software can extract. The processing time will vary depending on the selected type and amount of information to extract.

When a new data processing algorithm is developed to be added into the software, the following steps needed to be performed:

1. Decide on data channels needed and list those channels in the Supervisor class.
2. Set the message type and handling function in the Supervisor class for the new information extracted.
3. Add an entry in the output database structure for newly extracted information.

With these three steps, future researchers can add any number of data processing algorithms. Some of the important tasks the Supervisor class performs through different classes (fig. 3) are: 1) initialization of different data acquisition classes, 2) setup of the database structure, 3) message handling, and 4) extracted information output.

Case Study

Based on the architecture described above, software was developed to measure corn plant population and spacing with machine vision. Software development was done using Visual C++ (Microsoft, Redmond, WA) to perform batch processing of video.

ESCOPE performance

Video acquisition

Video was recorded in the field and then brought to the laboratory for analysis. Video data was transferred to the computer through an IEEE 1394 (fire wire) connection, whereas GPS was transferred using serial port. For spatial correlation, GPS data were saved along with video frame data. A commercially available VMS 200 unit (Red Hen systems Inc, Fort Collins, CO) was used to modulate the GPS signal and record it in the audio track of videotape. The VMS 200 unit was also used to demodulate and combine the video time code from the camcorder's LANC® serial output with the GPS signal.

Image processing

Image processing was accomplished in the ImageAnalyzer class. This class operated on the video frames stored in RAM. The Supervisor class provided a pointer to the DVIterator Class and the GPSIterator class to the ImageAnalyzer class. This class was designed to run on its own thread so that during real time analysis, the processing and capturing could be done in parallel.

The ImageAnalyzer class extracted the memory location of each video frame from the DVIterator class and mosaicked them to discard the overlapped amount of image from an image frame sequence. After sequencing the images, two features were extracted from each image: the total number of plant pixels, and their median position. Adjacent rows of the same class were grouped together and iteratively refined for final plant counting. The details of the ImageAnalyzer class are out of the scope of this discussion and are application dependent. However, the reusability of the class comes from the fact that the communication between the ImageAnalyzer class and the Supervisor class remains the same. ImageAnalyzer class not only detects the plants, but it also detects the field markers, and reports the detected plant center and marker center locations in terms of pixel count from the beginning of the image chunk to the Supervisor class.

To synchronize video images with GPS data, the video time code was used. The video time code was associated with each frame and also with the GPS signal. This provided a basis for geo-referencing each plant detected through image processing. However, the GPS signal update rate was much lower than the frame rate; hence the GPS locations for intermediate frames were linearly interpolated. Once the Supervisor class receives updates from the ImageAnalyzer class, it interpolates the location of each plant for its GPS location and writes to the output database table.

Information display and output

A plant-centric database format was used to output the information. In the plant-centric database, other data such as GPS location are interpolated for each detected plant. A plant-centric database is more suitable where information about individual plants is more important than statistics over a region like in the study of plant spacing and canopy area measurement.

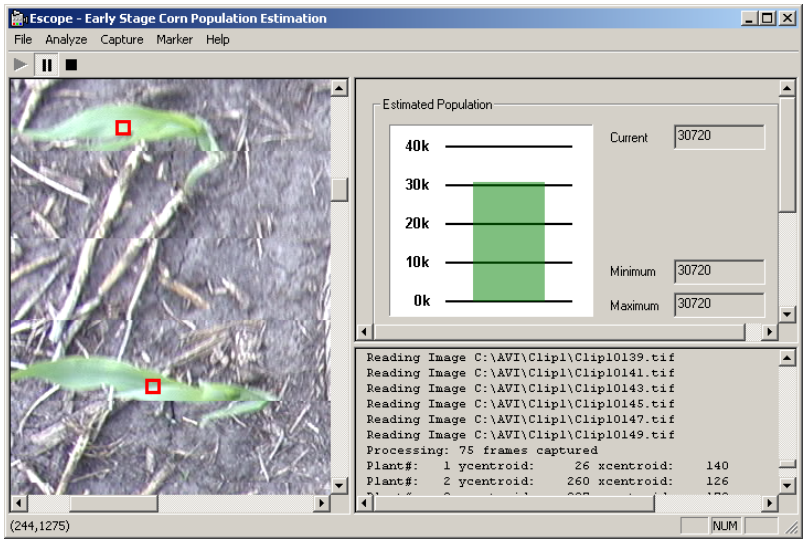

Figure 4. Interface of the program developed to estimate the plant location and spacing measurement. The program was names ESCOPE for early stage corn population estimation.

In order to display the extracted information, a View class was written. Once a chunk of an image is processed, depending on user preference, the Supervisor class sends a message to the View class to update itself and provides a pointer to the memory location of the database. Then the Supervisor class gets the data and plots the information on the computer screen. The View class produces three different windows panes to display different information. The first pane displays the chain of overlapped images (fig. 4). This window permits visual evaluation of image correspondence and plant detection performance. A red square is displayed over each detected plant location.

In the second window pane, plant population is displayed in the form of a bar graph. Maximum and minimum and current plant density measured in the field is also displayed (fig. 4). The third pane displays processing and other information in text format which is used during development. Finally, the program outputs a file structure with information for every plant detected (fig. 5). The first column in the database is the plant number. The second and third column show the plant location in terms of number of pixels from the beginning of a video chunk. The fourth column shows plot number. The fifth column shows the video time code when that particular plant was found. The last two columns are interpolated latitude and longitude of a plant.

Discussion

The object-oriented architecture provided a framework for developing video acquisition and image processing software. Direct Show® was successfully implemented for camera control and video acquisition. Use of the architecture led to more intelligent software development and efficient data flow and processing.

On average, for 60 seconds of video, ESCOPE took 43 seconds for image correspondence, and 5 seconds for feature extraction and plant counting on a computer with a 1.7 GHz Pentium-IV processor. A majority (91%) of the processing time was spent on image correspondence.

In order to demonstrate reusability of components, after coding the program for acquiring video signal from the camera, a feature was added to acquire images from the series of images stored in hard drive. A separate class called FileCapture was developed to read serial images and incorporated into the program. Only the Supervisor class needed to be modified to incorporate

the FileCapture class. The FileCapture class also used DirectShow® and has similar functionality as DVCapture except for camera control.

Plant#	x	y	Plot#	Timecode	Latitude	Longitude
1	126	602	1	00:09:26:14	42.01752513	-93.76642624
2	233	821	1	00:09:26:20	42.01752513	-93.76642823
3	254	1082	1	00:09:26:26	42.01752513	-93.76643022
4	306	1400	1	00:09:27:06	42.01752513	-93.76643354
5	259	1635	1	00:09:27:12	42.01752512	-93.76643553
6	581	2011	1	00:09:27:22	42.01752512	-93.76643885

Figure 5. Plant output of ESCOPE. First column shows plant detected. x and y shows plant position from the top of the row in pixels. Time code is the video time when plants were found. Latitude and Longitude are estimated GPS location of the plant.

To test the extensibility of the program, a class was derived from the ImageAnalyzer class with the functionality needed to calculate the top projected canopy area (TPCA) of each plant. The changes needed to accommodate this new feature were in the database structure in the Supervisor class and output format. The new class estimated the plant location in inches instead of the original pixel measurements. When the video frames were read from the hard drive using the FileCapture class and processed with the new ImageAnalyzer class, the output table contained different information than the original table (fig. 6).

Plant#	x(in)	y(in)	Plot#	Area(sq.in)	Image#
1	4.0	0.7	1	4.3	1
2	4.8	8.4	1	3.9	5
3	4.7	15.5	1	3.0	9
4	5.9	23.3	1	2.7	17
5	4.8	32.6	1	2.9	27

Figure 6. Output from modified ESCOPE. A FileCapture class read the images sequence from hard drive without GPS and new ImageAnalyzer calculated top projected canopy area of each plant.

Conclusion

An object-oriented architecture was developed and used successfully for a field data acquisition, processing and information extraction system. DirectX was used to control the camera, acquire video images and split the video images into individual images. The use of DirectX technology reduced the burden of taking care of every detail for hardware control and allowed the researchers to focus more on image processing and analysis. The architecture was reusable and extensible since none of the classes except for the Supervisor class needed to be modified to when incorporating a new image processing class.

Acknowledgements

This journal paper of the Iowa Agriculture and Home Economics Experiment Station, Ames, Iowa, Project No. 4003, was supported by Hatch Act and State of Iowa funds. Additional research support was provided by Pioneer Hi-Bred International Inc and the Iowa State University Center for Advanced Technology Development.

References

Aleman, J. L. F. and A. T. Alvarez. 2000. Can intuition become rigorous? Foundations for UML model verification tools. In *11th International Symposium on Software Reliability Engineering* 344-355. IEEE.

Beck, H. W., P. Papajorgji, R. Braga, C. Porter, and J. W. Jones. 2003. Object-oriented approach to crop modeling: concepts and issues. In *Proc. World Congress on Computers In Agriculture and Natural Resources* 297-305. F. Zazueta and J. Xin, eds. St. Joseph, Mich.: ASAE.

Birrell, S. J. and J. W. Hummel. 2000. Membrane selection and isfet configuration evaluation for soil nitrate sensing. *Transactions of the ASAE* 43(2): 197-206.

Birrell, S. J. and K. A. Sudduth. 1995. Corn population sensor for precision farming *ASAE Paper No. 951334*. St. Joseph, Mich.: ASAE.

Blackmore, S., S. Fountas, and H. Have. 2002. A proposed system architecture to enable behavioral control of an autonomous tractor. In *Proceedings of automation technology for off-road equipment* 13-23. Q.Zhang, eds. Chicago, IL. St. Joseph Mich.: ASAE.

El-Faki, M. S., N. Zhang, and D. E. Peterson. 2000. Weed detection using color machine vision. *Transactions of the ASAE* 43(6): 1969-1978.

Gary, C., J. W. Jones, and M. Tchamitchian. 1998. Crop modelling in horticulture: state of the art. *Scientia Horticulturae* 74(1-2): 3-20.

Goel, P. K., S. O. Prasher, R. M. Patel, J. A. Landry, R. B. Bonnell, and A. A. Viau. 2003. Classification of hyperspectral data by decision trees and artificial neural networks to identify weed stress and nitrogen status of corn. *Computers and Electronics in Agriculture* 39(2): 67-93.

Hummel, J. W., K. A. Sudduth, and S. E. Hollinger. 2001. Soil moisture and organic matter prediction of surface and subsurface soils using an NIR soil sensor. *Computers and Electronics in Agriculture* 32(2): 149-165.

Irmak, A., J. W. Jones, W. D. Batchlor, and J. O. Paz. 2001. Estimating spatially variable soil properties for application of crop models in precision farming. *Transactions of the ASAE* 44(5): 1343-1353.

Johnson, L. F., D. E. Roczen, S. K. Youkhana, R. R. Nemani, and D. F. Bosch. 2003. Mapping vineyard leaf area with multispectral satellite imagery. *Computers and Electronics in Agriculture* 38(1): 33-44.

Li, Z., P. P. Ling, and G. A. Giacomelli. 1998. Machine vision monitoring of plant growth and motion. *Life Support.Biosph.Sci.* 5(2): 263-270.

Lund, E. D., C. D. Christy, and P. E. Drummond. 2000. Using yield and soil electrical conductivity (EC) maps to derive crop production performance information. In *Proceedings of Fifth International Conference on Precision Agriculture,* P.C.Robert, R.H.Rust, and W.E.Larson, eds. Bloomington, MN, USA. Precision Agriculture Center, University Of Minnesota.

Morden, R. E., P. P. Ling, and G. A. Giacomelli. 1997. An automated plant growth monitoring system using machine vision. *ASAE Paper No. 974033*. St. Joseph, Mich.: ASAE.

MSDN Documentation. 2003a. CDocument. Microsoft Inc. Available at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcemfc/htm/cdocumnt_2.asp. Accessed on 29 May 2003.

MSDN Documentation. 2003b. Introduction to DirectShow. Microsoft Inc. Available at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/htm/introductiontodirectshow.asp. Accessed on 25 Apr. 2003.

MSDN Documentation. 2003c. The Component Object Model. Microsoft Inc. Available at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/com/htm/com_757w.asp. Accessed on 25 Apr. 2003.

National Research Council. 1997. Geospatial and information technologies in crop management. National Academy Press. Washington.

Nishiwaki, K., T. Togashi, K. Amaha, and K. Matsuo. 2001. Estimate crop position using template matching in rice production. *ASAE Paper no. 013103*. St. Joseph, Mich.: ASAE.

Object Management Group. 2003. Unified modeling language specification, Ver.1.5. OMG object management group. Available at: http://www.omg.org/technology/documents/formal/uml.htm. Accessed on 25 Apr. 2003.

Pan, X., J. D. Hesketh, and M. D. Huck. 2000. OWSimu: an object-oriented and Web-based simulator for plant growth. *Agricultural Systems* 63(1): 33-47.

Pierce, F. J. and P. Nowak. 1999. Aspects of precision agriculture. *Advances in Agronomy* 67: 1-85.

Root, M. D. and J. R. Boer. 1999. *DirectX Complete.* Ch. 1, 3-18. NY:McGraw-Hill.

Schmoldt, D. L. 2001. Precision agriculture and information technology. *Computers and Electronics in Agriculture* 30(1-3): 5-7.

Schueller, J. K., J. D. Whitney, T. A. Wheaton, and L. A. Balastreire. 2002. Accuracy and data characterization analysis in precision agriculture using matlab. *ASAE Paper No. 021044.* St. Joseph, Mich.: ASAE.

Shrestha, D. S. and B. L. Steward. 2003. Automatic corn plant population measurement using machine vision. *Transactions of the ASAE* 46(2): 559-565.

Sorensen, C. G., H. J. Olsen, A. P. Ravn, and P. Makowski. 2002. Planning and operation of an autonomous vehicle for weed inspection *ASAE Paper No. 021177.* St. Joseph, Mich.: ASAE.

Tang, L., L. Tian, and B. L. Steward. 2000. Color image segmentation with genetic algorithm for in field weed sensing. *Transactions of the ASAE* 43(4): 1019-1027.

Tarbell, K. A. and J. F. Reid. 1991. A computer vision system for characterizing corn growth and development. *Transactions of the ASAE* 34(5): 2245-2255.

Tarbell, K. A., D. K. Tcheng, and J. F. Reid. 1991. Corn growth and development attributes obtained using inductive learning techniques. *Transactions of the ASAE* 34(5): 2264-2271.

Torrie, M. W., D. L. Cripps, and J. P. Swensen. 2002. Joint architecture for unmanned ground systems(JAUGS) applied to autonomous agricultural vehicles. In *Proceedings of Automation Technology for Off-road Equipment Conference* 1-12. Q.Zhang, eds. Chicago, IL.

Van Henten, E. J. and J. Bontsema. 1995. Non-destructive crop measurements by image processing for crop growth control. *J.Agric.Engng Res.* 61: 97-105.

Will, J. D., J. F. Reid, N. Noguchi, and Q. Zhang. 2002. Software architecture design for automation of off-road equipment. In *Proceedings of Automation Technology for Off-road Equipment* 43-50. Q.Zhang, eds. Chicago, IL.

Zhang, N. Q., M. H. Wang, and N. Wang. 2002. Precision agriculture - a worldwide overview. *Computers and Electronics in Agriculture* 36(2-3): 113-132.