

DESIGNING A MACHINE-VISION SYSTEM

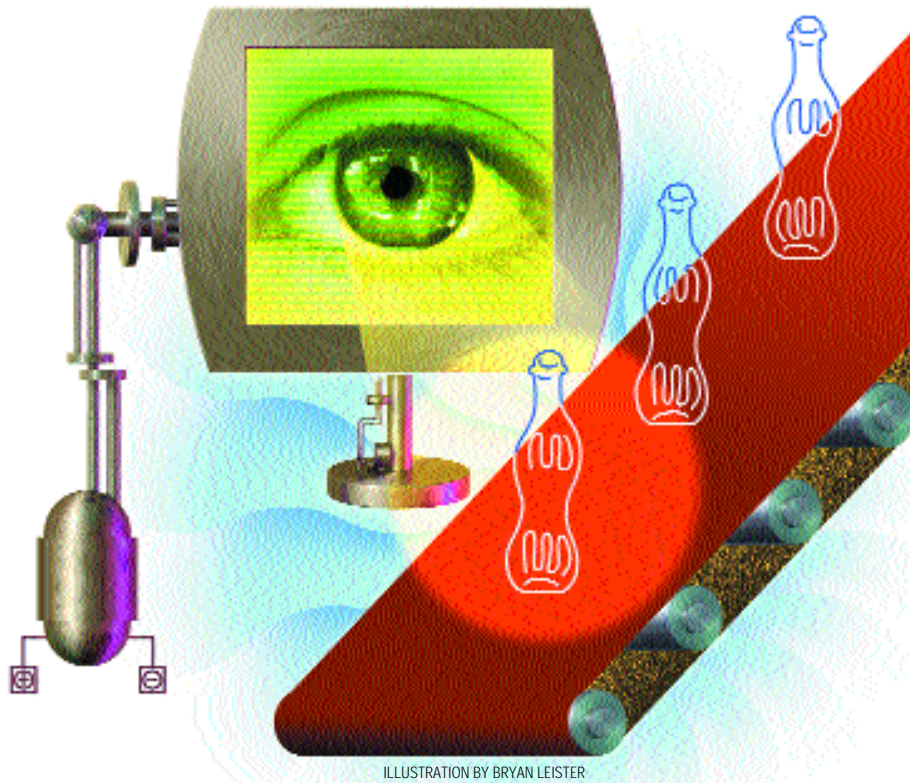


ILLUSTRATION BY BRYAN LEISTER

Knowing your goals and understanding your parameters up-front will yield improved vision system design.

By Christopher Poling, Data Translation

Modern machine-vision systems typically contain faster and more powerful PC platforms, robust 32-bit operating systems, and easy-to-use integrated software applications, making machine-vision systems more powerful, easier to program, and less expensive to use than ever before. To take full advantage of these powerful systems and painlessly integrate them into your manufacturing line, it is best to take some time to learn the basics about what makes up a vision system, how it is implemented, and the importance of proper planning.

Machine vision can be used in a wide variety of manufacturing operations for repetitive inspection tasks in which accuracy and reliability are important (e.g., verifying date codes on food packaging, inspecting automotive parts for proper assembly, and performing robotic guidance for pick and place operations).

building the system

Because the uses of machine vision are so diverse, specific components can vary from system to system. However, most systems generally include an input source, optics, lighting, a part sensor, a frame grabber, a PC platform, inspection

software, digital I/O and a network connection, and an X-Y positioning table (see figure).

The input sources and optics usually consist of one or more cameras and optical systems that take one or more images of the part being inspected. Depending on the application, the cameras can be standard monochrome RS-170/CCIR, composite color (Y/C), RGB color, non-standard monochrome (variable-scan), progressive-scan, line-scan, or custom CCD arrays (used for x-ray).

Illuminating the part for optimal data acquisition requires outside lighting. These assemblies come in various shapes and sizes and are available in a variety of intensities. The most common lighting types are high-frequency fluorescent, LED, incandescent, and quartz-halogen fiber-optic.

For maximum efficiency, the system must be triggered by a part sensor to acquire an image when the part under test is in the correct position. Often in the form of a light barrier or sensor, the part sensor sends a trigger signal when it senses that a part is in close proximity.

The frame grabber, or video capture card, interfaces the imaging units to the host computer. The frame grabber takes the image data provided by the camera(s) in either analog or digital

form and converts it to information for use by the host PC. This component is usually in the form of a plug-in board installed in the PC. A frame grabber can also provide signals to control camera parameters such as triggering, exposure/integration time, and shutter speed. Frame grabbers come in various configurations to support different camera types as well as different computer bus platforms (PCI, compact PCI, PC104, ISA, etc.)

A computer is a necessary part of a machine-vision system. Inspection applications generally require a Pentium III or equivalent. In general, the faster the PC, the less time the vision system will need to process each image. The vibration, dust, and heat often found in manufacturing environments frequently require the use of an industrial-grade or ruggedized PC. Software processes incoming image data and makes pass/fail decisions. Machine-vision software can come in many different forms and can be single function (designed only for one purpose like LCD inspection, ball grid array inspection, alignment tasks, etc.) or multi-function (designed with a suite of capabilities including gauging, bar code reading, robot guidance, presence verification, and so on).

Once the system has acquired an image, that image and the resulting data may need to be accessed by other users/systems to control the manufacturing process, communicate pass/fail information to a database, or both. Usually, a digital I/O interface board and/or network card makes up the interfacing through which the machine-vision system communicates with other machines, systems, and databases.

The X-Y positioning table automates the process of acquiring images of multiple samples. The table moves a predetermined distance after each image is acquired to properly position the next object or specimen in relation to the camera. The majority of X-Y tables have rapid smooth movement that minimizes image distortion and virtually guarantees a clean image without the need for using a progressive scan frame grabber.

choosing your hardware

Careful planning and attention to detail will help ensure that your inspection system meets your application needs. Knowing

your goals is perhaps the most important step in the process. Inspection operations fall into several categories: performing measurements or gauging, recognizing and identifying specific features (pattern matching), reading characters or encoded (bar code) information, detecting the presence of an object or marking, comparing objects or matching an object to a template, and guiding a machine or robot.

The inspection process can contain one operation or many depending on the requirements and goals. First, you should identify what tests you need to adequately inspect the part (measurement, presence verification, or optical character recognition), as well as what type of defects you expect to occur. To help identify which are most important, create a weighted list of required and optional tests. Build a list that satisfies the main inspection criteria. You can add additional tests later, but keep in mind that more tests means more inspection time.

Knowing your speed requirements for testing is critical. Knowing the amount of time the system will have to inspect each component or part will not only determine the minimum clock speed of the PC but may also affect the speed of the line. Many machine-vision software packages incorporate a clock/timer so that each step of the inspection operation can be closely monitored. From this data, you can modify the program and/or the motion process of the part to fit within the required timing window. Often, PC-based machine-vision systems can inspect 20 to 25 components per second, depending on the number of measurements or operations required and the speed of the PC used.

A machine-vision system is only as strong as its individual components. Any shortcuts made during the selection process—especially those involved in the optics and imaging path—can greatly reduce the effectiveness of a system. The following are a few basics you should keep in mind when choosing components involved in the image path.

Camera selection is directly tied to the application requirements and usually involves three main criteria: whether the test involves a color-specific imager, whether the parts will be in

motion, and what resolution will be required. Monochrome cameras are used for a majority of inspection applications because monochrome images provide 90% of the available visual data and are less expensive than their color counterparts. Color cameras are used when inspection applications require color-specific image data to be analyzed. Whether the part being inspected is stationary or in motion will dictate the exposure time and whether a standard interlaced camera can be used or if a progressive-scan camera is required to ensure a sharp image. In addition, the camera's resolution should be

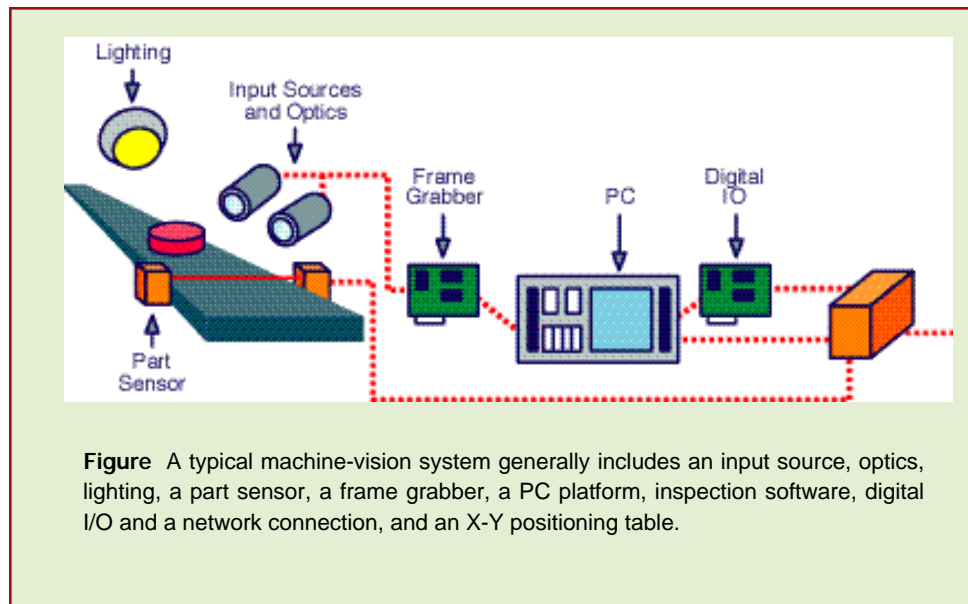


Figure A typical machine-vision system generally includes an input source, optics, lighting, a part sensor, a frame grabber, a PC platform, inspection software, digital I/O and a network connection, and an X-Y positioning table.

high enough to ensure that it can capture the proper amount of information needed for the inspection task. Finally, cameras should be high quality and rugged enough to withstand vibration, dirt, and heat present in an industrial environment.

Optics and lighting are often overlooked. When poor optics or lighting is used, even the best machine-vision system will not perform as well as a less capable system with good optics and adequate lighting. Design properly from the beginning, and don't depend on software to overcome design flaws. The typical goal for optics is to use lenses that produce the sharpest clarity and largest usable image, thus providing the best image resolution. The goal for lighting is to illuminate the key features being measured or inspected. The type of light used often will be dictated by those features, which include the color, texture, size, shape, and reflectivity of the object.

Although the frame grabber is only one part of a complete machine-vision system, it plays a very important role. The choice of frame grabber is defined by the characteristics of the camera it must interface to—for example, is it monochrome, color, digital, analog, and so on. With digital frame grabbers, the goal is to ensure that the digital image data from the camera is formatted properly prior to passing it onto the PC for processing. With analog frame grabbers, the goal is to acquire the image data from the camera and convert it to digital data with as little alteration to the image data as possible.

Using the wrong frame grabber can introduce errors in the image data. Industrial frame grabbers are typically used for inspection tasks, for example, while multimedia frame grabbers should be avoided. Multimedia boards can alter the image data with automatic gain controls, edge sharpening, and color-enhancement circuits. Although the images look more appealing to the eye, when processed by the system's software they can result in errors that directly affect the accuracy of the inspection process.

Removing as many variables as possible is key. The human eye and brain can identify objects in a wide variety of conditions. A machine-vision system is not as versatile; it can only do what it has been programmed to do. Knowing what the system can and cannot see will help you avoid false failures (wrongly identifying good parts as bad) or other inspection errors. Common variables to consider include ambient lighting, background color, requirement for image focus, and large changes in part color, finish, orientation, or position. Proper camera mounting, secure lighting positions, constant and repeatable part/component positioning, and blocking of external or surrounding lighting can eliminate many common set-up and false-failure problems.

New advanced algorithm technologies for dynamic machine-vision applications are emerging that can perform inspection applications without the need for strict control of operating parameters. These algorithms are designed to allow the system to function efficiently in environments that dynamically change. Machine-vision systems that use these new algorithms are no longer hindered by changes in part orientation, rotation, scaling, lighting, image quality, and so on. A machine-vision package with dynamic algorithms, such as our DT Vision Foundry package, can cope with changing parameters. For example, a search tool based on this technology is well suited to applications in which lighting variations, such as poor contrast, glare, reflection, and inversion, are problematic.

choosing your software

The machine-vision software is the centerpiece of the inspection system. The software selected will determine the length of time required to generate and debug inspection programs, what inspection operations can be performed, how well those operations can be performed, as well as many other important factors (see sidebar).

Machine-vision software packages that provide a graphical user interface are usually easier to program than their code (Visual BASIC or Visual C++) counterparts but can sometimes be limited when specialized features or functions are required. Although they require programming expertise, code-based packages can be more flexible if you're developing complex application-specific inspection algorithms. Some machine-vision software packages offer both graphical and code-level development environments, providing the best of both worlds and giving users the additional flexibility of selecting the environment needed to match the application requirements as well as the programming expertise available.

The overall objective for a machine-vision system is to perform a quality assurance role by separating the good parts from the bad ones. To do this, the system needs to communicate to the manufacturing line that a part is bad so that action can be taken. Usually this information is conveyed via the digital I/O board, which is connected to the manufacturing line's programmable logic controller (PLC). The bad part is then separated from the good parts. In addition, the machine-vision system may be connected to a network to allow data to be transferred to a database for data logging purposes. Data can then be analyzed by quality-control personnel to determine why the fault occurred.

Careful planning at this stage will ensure smooth integration of the machine-vision system into the line. Many companies would like to network their vision systems so that remote access is available. Transferring images and data can be very beneficial to the customer to prove total quality control. This information/requirement is very strategic to the design of a machine-vision system. You must know what PLC will be used and how it is interfaced, what types of output signals you'll require, what kind of network is currently used or required, and what kind of file formats will be transferred on the network. Typically, communication to a database is done via an RS-232 line connected to the manufacturer's network to track failure information.

When selecting components for a machine-vision system, consider future production requirements and changes. This can directly impact which features will be needed in the machine-vision software/hardware, as well as how easily the system can be altered to meet changing requirements and different tasks. Planning ahead will not only save time but will help reduce the overall cost of the system if it can be used for other inspection tasks in the future. **oe**

Christopher Poling is senior machine vision applications engineer at Data Translation, Marlboro, MA. Phone: 508-481-3700; e-mail: cpoling@datx.com.

software engineering: THE SEVEN DEADLY SINS

By Phillip Laplante, Penn State University

Perhaps because many imaging engineers are not trained in software engineering, or because of pressures to complete the project, basic software engineering practices are often not followed or followed poorly during imaging systems software development. Typical problems include lack of software requirements, poorly written requirements, failure to design for test, poor design of software, and improper or insufficient testing and documentation. Managers, engineers, and even customers often excuse these practices by citing pressures to market, high cost-to-benefit ratio, and (unfulfilled) promises to go back and fix things later.

Poor software-engineering practices early in the project can plague the system long after it is commercialized, costing time, money, and reputation. Therefore, making an early commitment to good software-engineering practices can pay huge dividends throughout the software life cycle.

the sins

My experiences in developing systems and consulting with other software engineers over the years have caused me to realize that there are several thematic problems that occur during the course of software development. Whether this situation exists because of poor training of the engineers, management, and customer pressures or simply a culture of malaise is unclear. It could be that because everyone is doing it, everyone goes along. In any case, these poor software engineering practices can be roughly organized into what I call "The Seven Deadly Sins of Software Development."

Pride

Failure to document code is one form of excessive pride that is manifest throughout the industry. "My code is self-documenting" is a familiar protest, along with "I had to work hard to develop the algorithm; therefore, others should work just as hard to understand it." The reality is that it simply isn't possible to make a nontrivial algorithm (such as those we find in imaging applications) easily understandable to every reader of the code. Documentation is a must.

Envy

Every engineer has his/her idol and favorite code written by that person. Certainly, code reuse is a wonderful and economical practice when followed correctly. No one wants to rewrite a module that is thought to work perfectly, and many of us feel ill equipped to challenge the assumptions or the reputations of others. When software is reused indiscriminately and without proper testing and documentation, numerous problems can occur, and these are very hard to detect.

Lust

It is an engineer's nature to anticipate needs and to provide for them. When developing software, we often believe that more is better—we can always find a way to use these features later. "Gold plating," or lusting for unnecessary features, can lead to memory and time overloading problems and should be avoided.

Greed

Perhaps the greatest of these sins, greed, is committed because of our eagerness to bring the software to market. And of all the sins, this is the one that management is most likely to condone. Yet study after study and years of practical experience have shown that investing a modest amount of time early in development to focus on specifications, design, and documentation can save millions in life-cycle costs later.

Gluttony

Excess code and over-engineering of algorithms is the bane of most engineers. We want things not only to work but to be a monument of ingenuity. Remember that parsimony and elegance, without sacrificing clarity, are essential for the maintenance of a software system through personnel changes over a long period of time.

Sloth

Perhaps the worst example of software sloth is failure to test the software sufficiently or to test without documentation. Insufficient testing can foster problems that will emerge later in the life of the system when sections of the code are stressed by expert users. Failure to document testing procedures or to develop a coherent test plan can make it difficult or even impossible to test a system as new features are added later.

Anger

Believe it or not, engineering can get personal. Team members stop talking to one another or set about outright sabotage. This is a management problem, but after all, much of software engineering falls more rightly into the realm of management than engineering. Team imbalances or personality issues need to be identified and fixed immediately.

Phillip LaPlante is an associate professor of software engineering at Penn State University, West Chester, PA. Phone: 610-725-5314; fax: 610-889-1334; e-mail: plaplante@gv.psu.edu.

SPIE
member

My colleagues and I are continuing research into the practices of imaging engineers in software requirements specification. To learn more and to participate in a survey assessing the state of affairs, please visit www.personal.psu.edu/cjn6/survey.html. Participants in the survey can request the study results; we hope to publish these results in an appropriate journal soon.