Controllable Time

Image Processing Basics: Real-time

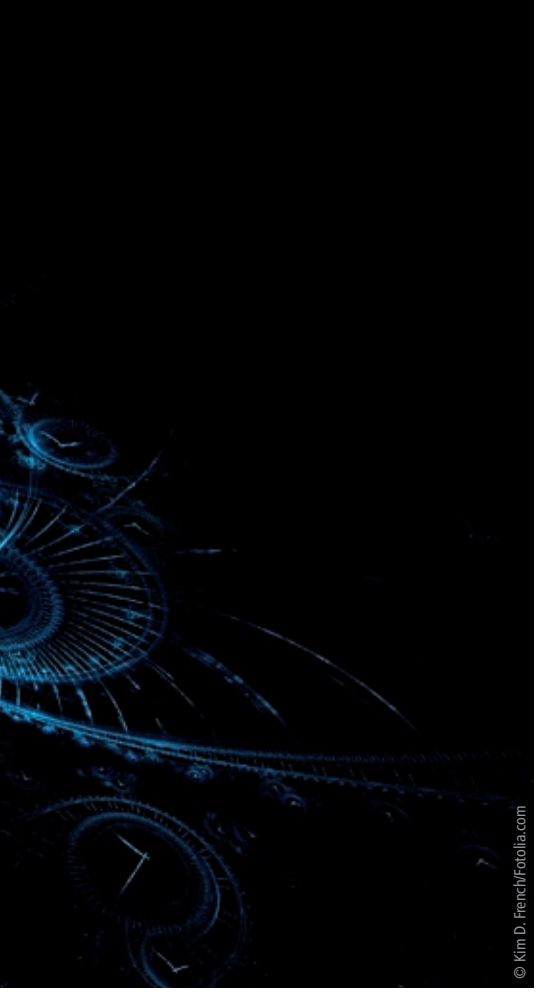
"Real-time" seems to be a cool term in conversations about machine vision, but raising the question about the precise meaning may trigger some stimulating discussions. "Real-time systems" often are simply assumed to be "fast", whereas some people use the term for image processing at the frame-rate of a consumer video stream. This article tries to give a definition of real-time processing within the framework of the inspection of moving objects in a production line. The main features are the ability of a real-time system to react to asynchronous sensor signals at any time and to call back within a defined time interval.

Interrupt

Moving parts in a production line will appear in the field of view of the inspection system usually not at a constant rate, but rather at random, asynchronously to any other process. An image thus has to be captured on demand within a certain well defined maximum time interval whenever an object to be inspected appears in front of the camera. In order to meet this requirement, any system suitable for this task shall be able to permanently read out a sensor signal (a light-barrier, e.g.) or to react to an external signal by starting the image capture and by triggering other peripheral components, if necessary, such as a strobe unit. The information extracted by the image processing routine then usually has to be sent back to the process or will be used to directly initiate some action downstream like a robot, e.g., picking a certain part. The sensor signal may appear at any time. Such an asynchronous signal is usually called an "interrupt request". The system used for inspection has to be able to immediately react to such requests and halt any other process which might currently run on the system. Since not a single object must pass the inspection zone without an image being captured and processed, the system status has to be changed from "idle" to "alert" within a defined maximum time interval whenever an object triggers the light barrier. It is by no means trivial to ask for this specific requirement.

When an interrupt source in an operating system has a sufficiently high priority, the usual process running in the system will be interrupted, and the interrupt service routine will be called as soon as the request has been acknowledged by the system. The interrupt service routine will then take full control of the system. When this task is finished, the system status has to be restored, and the usual routine will resume control. The status of the process thus has to be stored before the interrupt service routine takes over. This is quite similar to a common function call, but with the notable difference that a function is called at a precisely defined line of code whereas an interrupt request will appear at random. During the interrupt service routine further interrupt requests may appear which may try to cut in on the current process. It is immediately clear that interrupt handling is by no means trivial.

An inspection system in machine vision thus will perform image capture and image processing within the interrupt service routine. Looking at the general structure, these tasks are rather the exception, the usual process being a more or less idle cycle. This may seem to be a quite uncommon view of the problem, but just lean back and think: the system usually just waits in a loop for the next part coming along like an eagle circling in the air, always carefully looking at the sensor which will detect an incoming object, triggering the interrupt service routine, which causes the system to swoop



down and catch an image. While being idle, the system may well perform some useful tasks such as checking the lighting system or compensating the noise floor. But the routine doing the crucial job will be the interrupt service routine, which will call back to the process with the result of the image processing operations. The system may be busy with image processing in a situation where a large number of parts come along with a small distance between two subsequent objects, but it may also be idle for several seconds when a huge gap appears.

The image processing in an inspection system for a continuous production line thus works on demand: the event "light barrier detects a part" triggers the image capture and the image processing routine. The event may appear at random, at any time, asynchronously to any other process in the system. The system thus has to be able to detect an interrupt request and to finish the interrupt service routine under all possible circumstances which may occur during the operation of the system.

Keeping Pace

An interrupt request appears at random: the program may be working at a line of code somewhere in a function or at the beginning of the main procedure. Furthermore, since interrupt sources can only be scanned with a defined frequency, there will always remain an uncertainty about whether the incidence appeared at the beginning, at the end or sometime in between the time interval between the last and the last but one check on the interrupt flag. Managing interrupts is not trivial. During the interrupt service routine a further interrupt may be requested by the light barrier depending on how the signals at the sensor are evaluated. A large object travelling through may well trigger the interrupt again and again while blocking the light path. As an alternative, the light barrier might be programmed to trigger an interrupt when the object leaves the sensor rather than when it blocks the detector signal. Working with a system which is able to store all the interrupts coming in may also be a useful procedure. Even simple microcontrollers usually have several inputs to detect and latch interrupts.

Since an asynchronous hardware-interrupt always can only be detected with a remaining uncertainty in time, the position of the parts to be inspected will vary from image to image. Sensors and AD-converters also respond with a certain time lag and may show jitter. Image capture and usually a strobe-lighting must be triggered with a defined delay with regard to the sensor signal in order to catch the object precisely within the field of view of the camera. The call-back to the system also has to work in a well-defined manner on the time line to allow handling systems downstream to catch the proper object. The timeline of the events in reality thus must be mapped by the inspection routine in a sufficiently precise way to allow for tracking of the objects by all the mechanical and electrical components of the system in pace with the production cycle. The performance of the system according to this requirement is not only determined by the operating frequency of the processor, but also by the response times of the other hardware components.

Real-time Processing

A real-time system must be able to detect an asynchronous interrupt request, to halt the actual task of the program and to finish the interrupt service routine, whatever the status of the system may be when the interrupt request appears. Mapping of the real timeline, however, is not yet ensured by these requirements. In addition, an upper limit for the reaction time of the whole system, including image capture, image processing and call-back to the process, must be accomplished. The standard configuration must be re-established after a certain, well-defined time interval beginning with the event which triggers the interrupt request, in our case an object entering or leaving the light-barrier. Systems, which work according to this requirement such that a maximum reaction time can be guaranteed under all possible circumstances in the process, are called predictable or deterministic. This is quite a tough requirement – it means that a guaranteed deadline always, without a single exception, will be met.

The reaction time is the sum of the following time intervals:

- The time interval needed to process and evaluate the sensor signals to raise an interrupt request. Time constants of analog electronic circuits, gates and memory access enter into this time budget.
- The time interval needed by the operating system to detect an interrupt request.
- The time interval needed by the operating system to call the interrupt service routine. Several operating systems give higher priority to other, internal processes and ignore external interrupt requests when system resources are scarce.
- The time interval needed to finish the image processing routine including image capture.

The sum of the first three time intervals usually is called the interrupt latency. During this time interval the interrupt is present in the system, but has to wait for being acknowledged and serviced. Data sheets and application notes usually quote this time interval. Unfortunately, the interrupt latency for a given system is not a constant, but is distributed somehow. Therefore, you may find so-called typical data, sometimes the maximum of the distribution will be given, and to see the full distribution, measured within a defined scenario, will be a quite rare experience. Unfortunately, only the full distribution is a reliable basis for a decision about whether the risk related to the appearance of reaction times longer than the desired time interval can be taken or not. The forth component, however, should not be underemphasized: proper or sloppy programming of the image processing routine may have a tremendous influence upon the real-time performance of your system. An image processing algorithm may need more or less time to run through depending upon the precise content of the image. Classification, e.g., may branch into several different loops with significantly different processing times. Such a behaviour may be caused by iterations, recursion or undersampling with subsequent refinement, to name only a few possibilities. The performance should thus be carefully evaluated for any possible status of the program whenever the real time behaviour of the system might be compromised by the program module. When programs become complex to a degree where systematic testing is no option, real-time performance can no longer be demonstrated in a strict sense. Critical items in this context are recursions, which may oc-

cur when finding the roots of a system of equations or in interpolation, and the permanent availability of sufficient memory. Needless to say, a function from an image processing library can never be systematically tested, in a strict sense, by a user without access to the source code.

System Failure

A real-time system has to react under any possible external conditions within a defined maximum time interval, calling back with a deterministic result. Any reaction after the deadline will be regarded as system failure. As a consequence, the highest priority in the system will usually be given to the interrupt source, even higher than all priorities related to the internal processes of the operating system itself. Several well-established operating systems can not quote to be real-time systems according to this criterion, but need further modification by real-time extensions. That is a somewhat risky approach, since an operating system programmer aiming at office applications will probably not keep in mind the requirements of an extension which dares to tinker with his precious priorities. But never mind, there are derivatives based on common operation systems which have been developed precisely for real-time applications and are reported to work well. Peripheral components, however, may also compromise the real-time performance of a system. A classical strobe lamp, e.g., will fire at a rate basically determined by the time constant of the discharge capacitor. The real-time performance of the operating system may be first-rate in this scenario, but triggering the next strobe too early (because the next part already appears in the field of view) and without the capacitor fully charged, will usually not yield an acceptable image. In general, real-time performance is accomplished by systems which can capture an image on demand whenever an external signal triggers the process, react within a well-defined time interval by finishing the image processing routine, and call back to the process such that a deadline for the action to be taken can be guaranteed under any possible circumstances. Rather than throughput, availability of the processes and deterministic behaviour are the crucial issues in real-time applications. The acceptable maximum reaction time needed to keep pace with the production process, however, depends upon the requirements of the specific application. Since frame rates of 100 per second and transport velocities of 10 m/s are at the upper end of the requirement range for machine vision, reaction time intervals in the order of milliseconds usually will be sufficient to provide real-time performance, as long as parts come along one by one and with a specified minimum distance to each other. Real-time requirements for signal processing in airbags or ABS-brakes and in a lot of industrial control applications are much more demanding with reaction times in the microsecond range. With regard to real-time performance, image processing for inspection of moving parts in production lines can well be mastered with current technologies and will remain a safe field for a lot of years to come.

▶ **Author**
Prof. Dr. Christoph Heckenkamp
Darmstadt University of Applied Sciences
Department of Optical Technology and Machine Vision
heckenkamp@h-da.de
www.fbmn.h-da.de