Real-Time Imaging **8**, 157–172 (2002)
doi:10.1006/rtim.2001.0272, available online at http://www.idealibrary.com on IDEAL®

Real-Time Implementation of Obstacle Detection Algorithms on a Datacube MaxPCI Architecture

The high-speed civil transport (HSCT) aircraft has been designed with limited cockpit visibility. To handle this, the National Aeronautics and Space Administration (NASA) has proposed an external visibility system (XVS) to aid pilots in overcoming this lack of visibility. XVS obtains video images using high-resolution cameras mounted on and directed outside the aircraft. Images captured by the XVS enable automatic computer analysis in real-time, and thereby alert pilots about potential flight path hazards. Thus, the system is useful in helping pilots avoid air collisions. In this study, a system was configured to capture image sequences from an on-board high-resolution digital camera at a live video rate, record the images into a high-speed disk array through a fiber channel, and process the images using a Datacube MaxPCI machine with multiple pipelined processors to perform real-time obstacle detection. In this paper, we describe the design, implementation, and evaluation of this computer vision system. Using this system, real-time obstacle detection was performed and digital image data were obtained successfully in flight tests conducted at NASA Langley Research Center in January and September 1999. The system is described in detail so that other researchers can easily replicate the work.

Mau-Tsuen Yang[1], Tarak Gandhi[2], Rangachar Kasturi[2], Lee Coraor[2], Octavia Camps[2] and Jeffrey McCandless[3]

[1]*Computer Science & Information Engineering, National Dong Hwa University, Taiwan*
E-mail: mtyang@mail.ndhu.edu.tw
[2]*Computer Science & Engineering, Pennsylvania State University, USA*
E-mail: tarak_gandhi@hotmail.com. {kasturi,coraor,camps}@cse.psu.edu
[3]*Human Information Processing Research Branch, NASA Ames research Center, USA*
E-mail: jmccandless@mail.arc.nasa.gov

Introduction

Continued advances in the fields of image processing and computer vision have increased the interest in their ability to aid pilots in detecting possible obstacles in their flight paths. Acknowledging that the design of the high-speed civil transport aircraft (HSCT) has a limited cockpit visibility, the National Aeronautics and Space Administration (NASA) has proposed an external visibility system (XVS) in which high-resolution video images are obtained using cameras mounted on the aircraft. This system processes video images taken with

1077-2014/02/$35.00

an on-board camera directed outside the aircraft. The goal is to use computer vision algorithms to detect other aircrafts in the sky by analyzing the images captured by the on-board camera. The system is useful to help pilots make decisions and avoid air collision.

This system is not designed to be a replacement of radar sensors that are usually used for obstacle detection in the flight path. It just provides an additional means to detect an obstacle using visible light. The advantage to a radar system is that it can detect targets under poor visibility conditions caused by cloud cover or darkness. The drawback to a radar system is that the scan rate is typically only 45°/s. As a result, the radar system requires a fairly long time to cover a limited area. For example, if a radar system produces a beam of $3^\circ \times 3^\circ$ to scan for targets, 0.6 s are required to scan a $9^\circ \times 9^\circ$ region. Although the image-processing system is restricted to conditions of good visibility, it can process a $9^\circ \times 9^\circ$ region in about 0.06 s, which is substantially faster than the radar system.

Obstacle detection using image processing requires robust, reliable, and fast techniques. These techniques should provide a high probability of detection, while maintaining a low probability of false alarm in noisy, cluttered images of possible targets, exhibiting a wide range of complexities. The size of the image target can be quite small, from subpixel to a few pixels in size. As an example, consider a Cessna aircraft that has a length and a wing-span of approximately 9 m (30 ft) and the fuselage diameter of approximately 1.2 m (4 ft). The detection algorithm must be capable of detecting this small target at least 25 s prior to a possible collision to allow for corrective actions by the pilot. Assuming that both aircrafts are traveling at 125 m/s (250 knot), their relative velocity can be as high as 250 m/s (500 knot). In such a case, the aircrafts would be 6.25 km (3.5 nautical mile) apart 25 s before collision. Using a camera with a resolution of 120 pixels/$^\circ$, the image size of the aircraft is 10.0×1.4 pixels from a side view, but only 1.4×1.4 pixels from a front view. Furthermore, the detection algorithm must report such targets in a timely fashion, imposing severe constraints on their execution time. Finally, the system must not only work well under the controlled conditions found in a laboratory and with data closely matching the hypothesis used in the design process, but it must be insensitive – i.e., must be robust – to data uncertainty due to various sources, including sensor noise, weather conditions, and cluttered backgrounds.

Extensive work has been done on the problem of target detection. When the signal-to-noise ratio is low and the image motion of the object is small, it is preferable to use the "track before detect" approach. In this approach, an object is tracked over multiple frames before making a hard decision on the presence or absence of a target. The simplest way to integrate the input images over multiple frames is by temporally averaging them. However, when the object has a significant image motion, other approaches are required. Nishiguchi *et al.* [12] proposed the use of a recursive algorithm to integrate multiple frames while accounting for a small object motion. A dynamic programming approach was used by Barniv [2] and Arnold *et al.* [3] to detect moving objects of a small size. The theoretical performance of this approach was characterized by Tonissen and Evans [4].

There have been many real-time systems proposed in the past few years for object detection. Smith presented a real-time system called ASSET-2 [5] using custom hardware to detect and track moving objects against a moving background. Melton *et al.* [6] designed a real-time system using VLSI implementation to detect different types of objects over a range of image characteristics. Also, there have been many image-processing systems achieving high-performance computing using either application-specific integrated circuits (ASICs) [7] or field programmable gate array (FPGA) [8]. A major problem with these special purpose machines is that they require a low-level hardware design that is difficult to accomplish for developers of image-processing applications.

To avoid the inflexible and expensive process of hardware design, a Datacube MaxPCI system (shown in Figure 1) with multiple pipeline processors was used as the computing engine in this study. The Datacube MaxPCI system meets high-throughput low-latency demands and has been found useful by researchers working on real-time vision applications. A Datacube MaxPCI system was configured to capture image sequences from an on-board digital camera with $1\text{k} \times 1\text{k}$ resolution at a rate of 30 frames/s, record the images into a high-speed 64 GB disk array through a fiber channel, and process the images using multiple pipeline processors to perform real-time obstacle detection [17].

In this paper, we describe the design, implementation, and evaluation of such a computer vision system. Using this system, real-time obstacle detection was performed

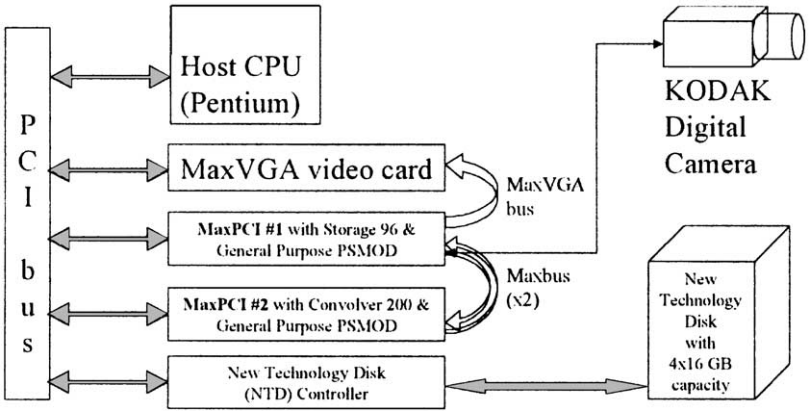

Figure 1. Overview of the real-time obstacle detection system consisting of a Pentium workstation, an NTD Recorder, a KODAK digital camera and two Datacube MaxPCIs image processing cards.

and digital image data were obtained successfully in recent flight tests conducted at NASA Langley Research Center. The image sequences, containing more than 100,000 image frames with $1\text{k} \times 1\text{k}$ resolution, were captured in the real flights with different pre-designed flight maneuvers. These image sequences have value to further research on obstacle detection algorithms under different conditions (size, contrast, background, etc.). Several real-time vision applications (e.g. visual robot navigation system, industrial parts inspection, and medical diagnosis) require rapid image analysis by computers. By discussing the key issues associated with this system, the time needed by others to implement similar real-time systems can be reduced.

The next section provides an overview of real-time image capturing, recording, and processing by the system. To follow the implementation issues regarding obstacle detection, algorithms on the MaxPCI system are dealt with. The aircraft maneuvers in the flight tests and results of the system's performance are then described. In the final section, conclusions are presented.

System for Real-Time Image Capturing, Recording, and Processing

This section provides an overview of real-time image capturing, recording, and processing by the system. The following sub-section describes real-time image capturing using a high-resolution digital camera. The second subsection explains real-time image recording using a high-speed disk array and the subsequent one deals with issues of real-time image processing using multiple pipeline processors.

Image capturing using a remote digital CCD camera and motorized lens

A critical component of the vision system is the imaging sensor. A Kodak Megaplus ES1.0 charge-coupled device (CCD) digital camera with a Cosmicar/Pentax 1 in (50 mm) motorized lens was chosen because digital CCD cameras offer superior performance compared to their analog counterparts [9]. Digital cameras are also highly immune to the spatial and temporal artifacts caused by transmission-line noise. The Kodak ESl.0 captures 30 frames/s with a $1\text{k} \times 1\text{k}$ resolution in an 8-bit format (256 gray levels) [10]. It was mounted in the cockpit of a modified Convair C-131 aircraft, called the total in-flight simulator (TIFS). Since the recording system was located in the aft portion of the aircraft, a l00-ft digital data cable transferred the image data signals to the recording system. The synchronized clock signals generated by the camera were also transferred through the data cable. A good-quality cable with a low capacitance prevents asynchronism and noise that can occur with lengthy cables.

The dynamic range of the captured images was very large due to variations in factors such as the sun orientation, cloud conditions, and aircraft altitude. To prevent saturation or very low gray levels in the captured images, a motorized aperture lens was installed on the camera and a remote aperture control box next to the recording system (100 ft from the camera). With this motorized aperture, the operator manually adjusted the

aperture before the flight to prevent extremely bright or dark images. No operator interaction is required during the flight. The camera exposure control software provided by Kodak was not used because it could inadvertently produce blurred images (caused by extended exposures) or unacceptable noise levels (caused by brief exposures).

Real-time recording of digital image sequences using new technology disk (NTD)

A typical flight sequence with a target aircraft in the field-of-view can last several minutes and produce thousands of $1\,k \times 1\,k$ images. One means of reducing the massive amount of disk storage space needed to hold these images is compressive algorithms. However, because of the desirability of analyzing the camera's raw characteristics, uncompressed images were stored. The task described in this paper requires a system recording data at a rate of 30 MB/s (or 1.8 GB/min).

To satisfy these large bandwidth and storage requirements, a Pentium 233 workstation (running Windows NT) with two internal MaxPCI cards from Datacube, Inc., and an external disk array, called the new technology disk (NTD), were used. The MaxPCI is a real-time image-processing card with pipeline image-processors that provide a cost-effective way to meet high-throughput, low-latency demands. The data cable was connected from the digital camera to one of the MaxPCI cards. The camera sends images through two channels, with odd lines in one channel and even lines in the other channel. The MaxPCI card receives the images through these two channels, with a throughput of 15 MB/s from each channel. The MaxPCI card was configured to merge the two channels in order to form complete images in the MaxPCI's memory. The images are then sent via the MaxVGA bus to the MaxVGA card for display, and via the PCI bus to the Adaptec AIC-1160 disk controller card for storing. Transfer to the disk controller card was accomplished using high-speed image access (HSIA), a technique that moves data directly back and forth between the disk controller card and the MaxPCI's memory, without being copied in an intermediate memory buffer. This eliminates copying data to the host memory, as would be required by other disk storage products. Finally, the images are transferred through a fiber channel (FC) cable to the NTD, where they are stored. The FC transmits data between computer devices at a rate of up to 1 GB/s. Since it is three times faster than the small system computer interface (SCSI), FC is expected to replace SCSI as the transmission interface between servers and storage devices. The NTD is a redundant array of independent disks (RAIDS) subsystem that enables high-speed lossless digital image recording and playback. The NTD used was a four-disk array with 16 GB per disk. With the FC option, it is possible to achieve NTD transports in excess of 32 MB/s. To achieve the highest access speed, there is no formatting of data storage on the NTD – all images are recorded as plain raw data to the NTD's consecutive physical sectors. The NTD can record and playback images at a real-time frame rate of up to 40 MB/s.

NtdIfLib, which stands for NTD ImageFlow Library, is a C-callable library for programmers to use in creating their own NTD access programs. The NtdIfLib is integrated with ImageFlow, a C-callable library that configures and manages data transfers on the MaxPCI to perform real-time image processing. With the power of the NtdIfLib and ImageFlow programming, the system not only can record the digital images in real-time, but it also can be extended to perform several image-processing algorithms concurrently. Developing a parallel program on the MaxPCI is complicated; the programmer must know a good deal about the underlying hardware. Moreover, since there is no useful debug tool for ImageFlow at this time, the programming task using ImageFlow is time-consuming. However, a very satisfactory system can be developed with appropriate effort.

Real-time image processing using MaxPCI image-processing cards

The MaxPCI is a real-time image-processing card with pipeline processors manufactured by Datacube Inc. It uses a Windows NT workstation as a host machine and supports multiple simultaneous pipelines that can be switched by software at read-time frame rates. Our Datacube IP system is equipped with two MaxPCI IP cards, each of which consists of five modular hardware devices and a set of memories connected by a large programmable switch (shown in Figure 2). The five devices are (1) the MaxAcq acquisition unit that receives either a digital or an analog signal from the camera; (2) the MaxVGA display unit that outputs the video signal to the MaxVGA video card; (3) the arithmetic unit (AU) which performs arithmetic and logical operations; (4) the look-up table (LUT) that performs pixel value transformations; and (5) the advanced memories (AMs) component which can receive an image from the cross-point switch and transmit another image to the cross-

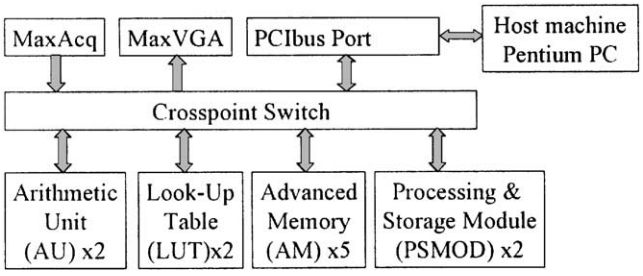

Figure 2. Architecture of a MaxPCI card. Each MaxPCI is composed of a MaxAcq acquisition unit, a MaxVGA display unit, five AMs, two AUs, two LUTs and two PSMOD add-on modules.

Table 1. The number of main resources in each MaxPCI card

	Abbreviation	Amount (ID)
MaxPCI #0 resource		
Arithmetic unit	AU	2 (0–1)
Advanced memory	AM	5 (0–4)
Look-up table	LUT	2 (0–1)
Delay element	DLY	2 (0–1)
General purpose add-on PSMOD	GP	4 (0–3)
Storage96 add-on PSMOD	ST	6 (0–5)
Analog acquisition module	QA	1
MaxPCI#1 resource		
Arithmetic unit	AU	2 (0–1)
Arithmetic memory	AM	5 (0–4)
Look-up table	LUT	2 (0–1)
Delay element	DLY	2 (0–1)
Convolver200 add-on PSMOD	VD	1
General purpose add-on PSMOD	GP	4 (0–3)
Digital acquisition module	QI	1

point switch at the same time. The AM also allows the host computer to read or write pixels via the PCI bus.

Moreover, each MaxPCI may be extended by the selection of two add-on processing and storage modules (PSMODs). The variety of the PSMODs enables users to balance their processing, memory and resource needs. Up to two PSMODs can be installed on each MaxPCI. The first MaxPCI in our system was equipped with a Storage96 (ST) and a general purpose (GP) PSMOD, while the second MaxPCI was equipped with a GP and a Convolver200 PSMOD. These devices operate on pixel arrays at 40 MHz. The MaxPCIs communicate with each other through two buses called Maxbuses, each of which has a bandwidth of 160 MB/s. The MaxVGA is a separate display card which inputs images from the MaxPCI through a private MaxVGA bus. Hence, the display can be accelerated without interfering with the PCI bus traffic. Table 1 lists the main resources in each MaxPCI card. It should be noted that ST is the same as AM, while GP is similar to AU. Another important constraint is the communication channels (CHs) between the two MaxPCI cards. There are only eight CH channels, with 8 bits in each channel. Thus, the traffic between two MaxPCIs should be minimized to preserve the precious CHs. The first MaxPCI was equipped with an analog acquisition module (OA) to input an analog signal, while the second MaxPCI was equipped with a digital acquisition module (QI) to input a digital signal from a camera. The next section provides the details and implementation issues for both looming and translating target detection algorithms.

Implementation of Obstacle-Detection Algorithms on MaxPCI

This section deals with the implementation of obstacle detection algorithms on the MaxPCI system. The first subsection explains the concept of pipeline scheduling. The second describes the important issues of MaxPCI programming. The subsequent subsection summarizes the obstacle detection algorithms for both looming and translating targets. The fourth presents the implementation of the detection algorithm for translation targets, while the implementation of the detection algorithm for looming targets is presented in the last subsection.

Pipeline scheduling

To execute an algorithm efficiently, the concurrency features in the algorithm should be exploited. The concurrency can be divided into two types: spatial and temporal. Spatial concurrency (parallelism) involves tasks that can be executed by several processors simultaneously, while in temporal concurrency (pipelining), chains of tasks can be divided into stages, with each stage handling results obtained from the previous stage. Spatial concurrency can be exploited to reduce the execution time of a single image frame. Temporal concurrency can be exploited to further increase throughput whenever a long sequence of image-processing tasks is applied on a continuous image sequence. In our project, we exploited both spatial and temporal concurrencies (parallelism and pipelining) present in the task graph. To avoid resource conflict in a pipeline schedule, a processing pipe cannot flow through the same resource more than once. Moreover, if several

pipes execute concurrently, none can share the same resource.

The design of pipeline scheduling can be divided into three steps: (1) partition the dependency graph into several pipeline stages; (2) allocate resources for each pipeline stage; and (3) schedule the operations inside each pipeline stage using allocated resources. Traditionally, a MaxPCI programmer must schedule an algorithm to the available processors manually so that the program is capable of running efficiently with several processing units. However, this approach is time-consuming and impractical in a system that must perform a variety of different algorithms and should incorporate new algorithms as they are developed. An automatic pipeline scheduler developed for this purpose is described in a separate report [11].

Important issues of MaxPCI programming

ImageFlow [12] is a low-level library of C-callable functions that configure and manage data transfers on the Datacube MaxPCI pipeline-processing devices. ImageFlow allows the programmer to specify connections between the processing elements, as well as between ports on the cross-point switch. It also provides access to attributes associated with each processing element. Programmers cannot simply state that two image streams are to be added; they must specify which ALU is to be used, where in memory the images are stored, and the path the images will take to reach the ALU. It is a programmer's job to handle resource conflicts. In our project, ImageFlow was used to develop parallel programs for all obstacle-detection algorithms.

An algorithm should be defined as multiple parallel pipes to accomplish the desired tasks efficiently. These algorithms are then mapped to a sequence of pipeline processing elements. Each ImageFlow program should have at least three pipes: acquisition, processing, and display. The acquisition pipe obtains image sequences from the camera, while the display pipe offers a stable output for the monitor. In our applications, handling the whole processing in one pipe is too complicated – thus, the processing is partitioned into many pipes.

To optimize the performance of our implementation, there is an important feature in ImageFlow programming called pipe altering thread (PAT) [12]. The use of PAT can reduce rearm time for a pipe, and is vital to efficient applications. PAT speeds up the image processing by pre-calculating the pipe delay and parameter setting. However, it also increases the complexity of the ImageFlow programming.

Obstacle-detection algorithms

We are interested in two types of targets (shown in Figure 3): translating targets (the target aircraft cross-perpendicular to the direction of the host aircraft); and looming targets (target aircraft flying away from (or near to) the host aircraft). The looming targets are more dangerous because they are on a collision course.

Over the past year, several algorithms were combined to form a composite system for the detection of looming targets [9]. The steps to this composite system are as follows:

(1) *Temporal Averaging*: For objects with a uniform background, having a very small image motion, such as those on a collision or near-collision course. When the target motion is small, temporal averaging improves the SNR and reduces the processing rate required for subsequent steps.
(2) *Pyramid construction with morphological filtering*: In this pre-processing step, a pyramid is constructed to accommodate different sizes and velocities of objects. Morphological filtering [13] is performed at each pyramid level to remove large background clutters due to clouds and/or ground to improve performance.
(3) Dynamic Programming: A dynamic programming algorithm [3] is performed on pre-processed frames to integrate the signal over a number of frames by taking the target motion into consideration.

It should be noted that one or more of these steps can be bypassed so that any of the basic algorithms described above can be tested individually using the same system.

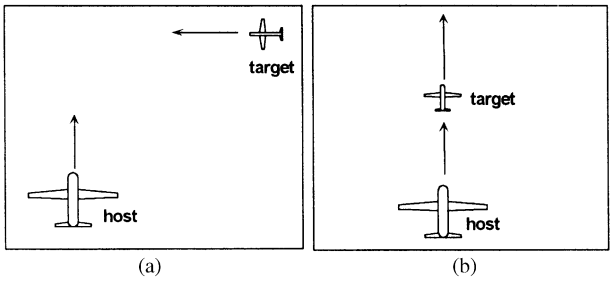

Figure 3. Two kinds of flight maneuvers: (a) Translating maneuver and (b) looming maneuver.

The above target-detection algorithms were implemented on the Datacube MaxPCI system.

In addition to detection of objects on a collision course, it is useful to monitor the objects crossing in front of the aircraft. For this purpose, a system was designed to specifically detect objects which have a translating motion in the image. To distinguish translating objects from ground or cloud clutter, the following criteria were used:

(1) The object should have sufficient signal strength.
(2) The object should have an image velocity greater than a threshold.
(3) The object should have a consistent motion.

The algorithm to detect translating objects also has been implemented on the Datacube MaxPCI system to obtain real-time performance. The system was mounted on the host flight aircraft and performed well in detecting and tracking objects. The algorithm is divided into two concurrent parts. These parts are (1) image-processing steps which remove most of the clutter, and isolate potential features which could be translating objects and (2) tracking these features using a Kalman filter to distinguish genuine translating objects from background clutter which was not separated by the simple image-processing steps of the first part.

In the first part, these steps consist of temporal differencing, low-stop filtering, non-maximum suppressing (NMS) and feature extraction. Image-processing operations are suitable for a pipeline architecture, and can be done in integer format. Hence, these steps are implemented on the Datacube MaxPCI machine. The output of this part is a list of image points which are likely to contain the target objects, along with their signal strengths. In the second part, since the first part has reduced the volume of data to be operated on, more complicated target-tracking algorithms can be implemented even on the host PC associated with Datacube. The threshold used in the first part is adjusted dynamically to give a nearly constant number of features for the second part so that they can be processed in real time using the slower host. This is known as the rate constraint [14].

Implementation of obstacle-detection algorithm for translating targets

The implementation of detection for translating targets can be divided into three subsystems: record/playback, image processing, and tracking. The first and second subsystems are implemented on Datacube MaxPCI cards, while the third subsystem is implemented on the host CPU. All three subsystems should be executed simultaneously to make the whole system run as fast as possible. Hence, care should be taken to avoid any resource conflict and reduce the communication between subsystems.

Real-time recording and playback from the NTD is useful in algorithm testing and development. Both portions of the record/playback subsystem need to handle double-buffering because the NTD operates at a faster rate than the MaxPCI. The HSIA port on the MaxPCI offers a good performance in accessing image data via the PCI bus. Basically, the implementation of playback is similar to recording, with a reverse order of pipeline connections, and is simpler because the camera is no longer necessary, allowing us to skip the acquisition pipe.

Image-processing subsystem for translating targets. This subsystem performs the basic image-processing steps to suppress clutter and extract features that could be translating targets. The image-processing subsystem is the most complicated subsystem and occupies the most resources. Figure 4 shows an overview of this subsystem. The first three steps are theoretically interchangeable, since they are all linear filters.

Temporal differencing. Image differencing was performed on the low-stop filtered images by subtracting consecutive frames. This is equivalent to a low-stop filter in temporal direction. Since the target aircraft was assumed to cross the host aircraft, the target aircraft has a significant translation in the image. On the other hand, the objects corresponding to background clutter are relatively stationary because their distances are usually much longer than the distance of the target aircraft. Image differencing suppresses stationary objects corresponding to background clutter. Two AM units (Am1_2 and Am1_3) were used as a temporal buffer to store the last two frames (shown in Figure 5). The difference image was obtained by subtracting the frame before the last frame from the current frame. Each arithmetic unit (AU), or general purpose (GP) unit, in the MaxPCI can be divided into separate linear (AU_Linear) and nonlinear (AU_Nonlinear) parts. The AU_Linear can be configured to perform any arithmetic operation, while the AU_Nonlinear can simultaneously perform any logical operation. Both the AU_Linear and AU_Nonlinear are capable of handling four inputs at one time. In this example, the AU_Linear section in a general

purpose unit (Gpl_1) was configured to perform a subtraction operation.

Subsampling. Subsampling was used to divide the image size by a factor of two, so that the system was capable of execution in real time. A low-pass filter was performed before down-sampling to reduce the loss of subpixel information. This step reduced the resolution of the image by two. However, since the size of the translating target was usually greater than 2 pixels, this step actually enhanced the signal-to-noise ratio due to the spatial integration performed by the low-pass filter. Figure 6 shows the implementation of a 5×5 Gaussian low-pass filter. A neighborhood multiply and accumulate unit (VD_NMAC_A) in the Convolver200 is configured to perform the low-pass convolution. Subsampling was performed by adjusting both horizontal and vertical zoom factors of the receiving gateway in an AM unit (AMl_4).

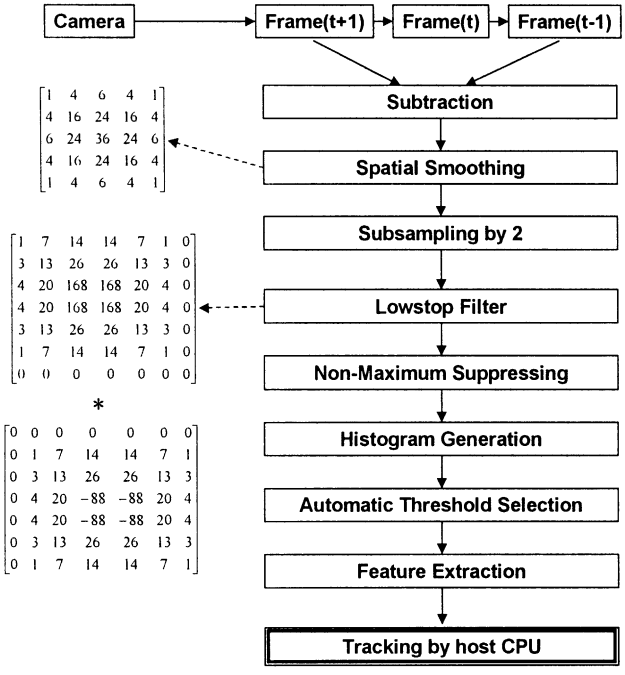

Figure 4. The temporal differencing algorithm for the detection of translating targets.

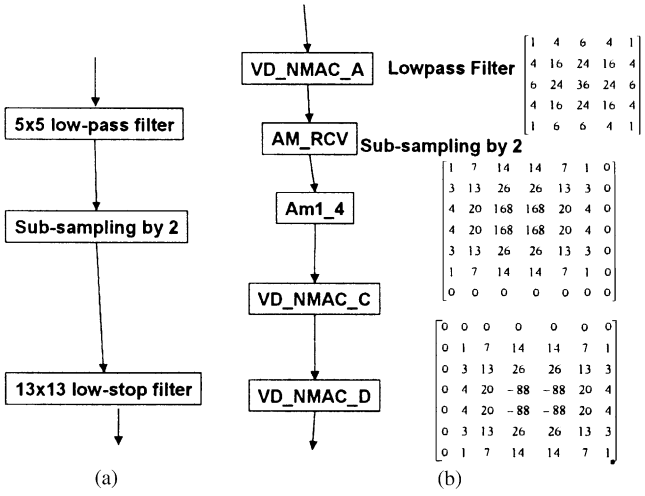

Figure 6. The implementation of sub-sampling and low-stop filter on Datacube MaxPCI. (a) Flowchart (b) Hardware mapping.

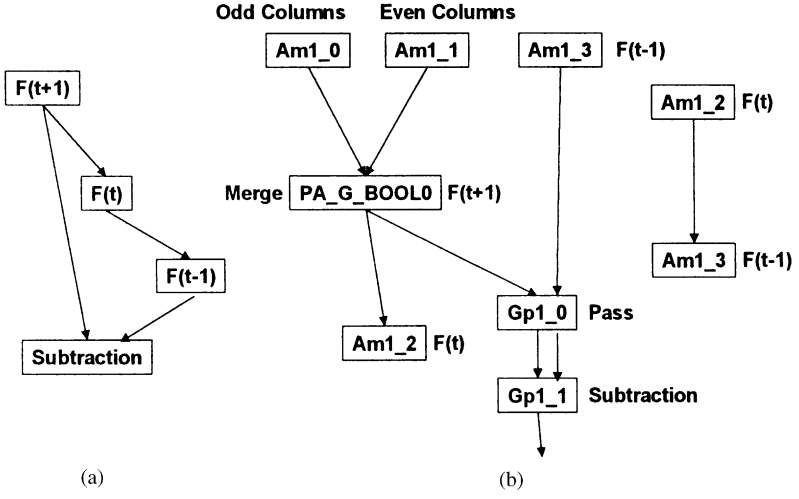

Figure 5. The implementation of a temporal differencing algorithm on Datacube MaxPCI. (a) Flowchart (b) Hardware mapping.

Low-stop filtering. A low-stop filter [15] was applied to the reduced image to suppress background clutter. The filter was formed by subtracting a large-sized low-pass filter from a small-sized low-pass filter (shown in Figure 6). The effect of a low-stop filter is similar to a high-pass filter in that low-frequency components are eliminated while high-frequency components are reserved. The filter mask is used. The filter effectively subtracts from each pixel the mean of its neighborhood, thereby suppressing uniform background intensity and weak clutter. Since a half set of Convolver200 resources was occupied by subsampling, only 100 points of the neighborhood multiply and accumulate units (VD_NMAC_C and VD_NMAC_D) in the Convolver200 can be used in this step. To use these 100 points efficiently, two sequential 7×7 low-stop filters were used to simulate a big 13×13 kernel low-stop filter.

Non-maximum-suppressing. Direct use of the output from previous steps would increase the number of features for an extended target. NMS was used to extract the local maximum, and to get a single feature (or sometimes a small number of features) for the entire target. If the magnitude of a pixel was not greater than all the neighbor magnitudes, then the magnitude of the pixel was set to zero. After applying NMS, the number of final features can be reduced and the overall throughput can be increased. A 3×3 NMS was implemented in three steps. First, a 3×3 dilation image was performed by replacing each pixel in the original image by the maximum of its 3×3 neighborhood. Second, the difference image between the original image and the 3×3 dilation image was obtained. A pixel in the difference image was zero if and only if the pixel was a 3×3 local maximum in the original image. Finally, the difference image was thresholded to extract those pixels that were equal to zero in the difference image (that were local maximum in the original image).

Pixels can have both positive or negative values corresponding to bright and dark targets, respectively. It is desired to detect both positive and negative targets. However, two sets of 3×3 dilation would consume a lot of resources, especially AUs. To save precious resources, an absolute operation was performed before NMS, so that only one set of dilation was required (shown in Figure 7). Every pixel that is not at a local maximum in its 3×3 neighborhood is marked. The marked pixels are set to zero in the original image. In this example, the nonlinear section in an AU (AUl_1) was configured to perform a maximum operation with three inputs. The three inputs form a 3×1 neighborhood window, i.e. (x, y), $(x+1,y)$, and $(x+2,y)$.

Histogram accumulation and automatic threshold selection. To extract candidate features, the image obtained from the above steps should be thresholded. Furthermore, the threshold should be chosen so that the number of features neither overloads the tracking subsystem, nor keeps it unnecessarily idle. Hence, the threshold was selected so that the number of pixels exceeding the threshold was less than or equal to a fixed rate that matches the operation speed of the tracking subsystem. For this purpose, a histogram of the image was constructed. The threshold then is determined as the smallest pixel value for which the number of elements in the histogram bins above this value does not exceed the

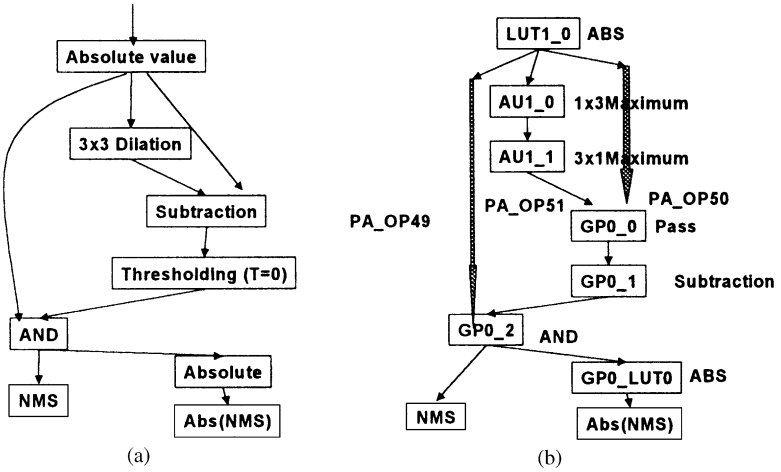

Figure 7. The implementation of a non-maximum suppressing on Datacube MaxPCI. (a) Flowchart (b) Hardware mapping.

fixed rate. Applying this value as the threshold ensures that the number of features remains bounded. A histogram processor (HP) in MaxPCI can generate a histogram consisting of 256 bins, with 2 B in each bin. The histogram was accumulated using an AU (AU0_1) to determine the threshold value. An LUT (LUT0_0) was configured according to the automatic selected threshold value. Finally, pixels in the image with an amplitude value smaller than the threshold value were suppressed using LUT0_0.

Feature extraction. The pixels in the image with an amplitude greater than the threshold are separated as features. The position and amplitude of each feature were transmitted to the tracking subsystem, with information extracted using the statistic module inside an AM. Feature coordinates were extracted using an AM unit (AMO_2), while feature amplitudes were extracted using another AM unit (AMO_3). Since the threshold value in the last step was selected so that the number of pixels exceeding the threshold would be less than a fixed rate, the number of features was guaranteed to fit into the memory surface safely. Since the tracking subsystem was implemented using the host CPU, the features were transferred from the MaxPCI to the host memory so the tracking subsystem could read from the host memory directly. The technique of HSIA was used to perform efficiently the feature transmission through the PCI bus.

Tracking subsystem. This subsystem maintained a list of tracks containing the frame number, a unique ID, position, velocity, and amplitude. The feature amplitude is defined as the magnitude of the feature pixel after the image-processing steps described in the prior section are applied. Refer to [16] for the detailed description and parameters of the tracking subsystem. In summary, the following steps were repeated for each frame:

(1) *Input*: The list of features was received from the image-processing subsystem.
(2) *Track update*: For each track in the list of tracks, the list of features was scanned to obtain features in a neighborhood window around the track's position. If there were any such features, then the strongest was selected as the continuation of the track. Using the coordinates of this feature, as well as current track position and velocity, the expected position and velocity for the next frame were estimated using a Kalman filter. If no such feature was found in the neighborhood of the track, position and velocity were extrapolated in the same Kalman filter framework, using only the current values for the track. The track amplitude was updated using recursive averaging with a forgetting factor:

$$F_{n+1} = f_n + \alpha F_n$$

Where F_n and F_{n+1} are the track amplitudes for the current and next frames. f_n is the feature amplitude. α is the forgetting factor.
The feature amplitude f_n is zero if no feature is found.
(3) *Formation of new tracks*: After all current tracks were updated, features in the feature list were used to check for new tracks. For every feature, the list of tracks was scanned to see if a track was already in its neighborhood. If not, a track was created out of the feature. Its position should be the same as the feature position, whereas velocity was initialized to zero. The actual velocity was computed only in the next frame.
(4) *Pruning the list of tracks*: If the number of tracks is too large, the subsystem can get overloaded and fail to operate in real time. To eliminate this possibility, if the number of tracks was greater than a particular number, the weakest tracks were deleted.
(5) *Merging similar tracks*: Two or more tracks may be formed corresponding to the same object. Hence, tracks that were very close to each other and had nearly the same velocity were merged, retaining the one with larger amplitude.
(6) *Output*: Tracks that satisfy object criteria, including amplitude larger than a threshold, and other factors were output as potential objects.

The tracking subsystem not only tracked targets moving horizontally across the image, but was generally enough to track any target moving in a consistent direction. In addition, the system was able to track several targets simultaneously since several tracks were maintained simultaneously. Table 2 summarizes the execution iteration period for the operations described in this section. The required resources are based on the implementations on the Datacube MaxPCI system. The reported execution iteration period is based on an input image with $1\,\mathrm{k} \times 1\,\mathrm{k}$ resolution.

Implementation of obstacle-detection algorithm for looming targets

The implementation of the detection algorithm for looming targets can be divided into two subsystems: record/playback and image processing. The record/playback subsystem is identical to the record/playback

Table 2. The execution response time for operations implemented on the Datacube MaxPCI system

	Iteration period (ms)
Operations for translating targets (input size)	
NTD record/playback (1 k × 1 k)	24.9
Camera acquire (1k × 1k)	34.1
Temporal differencing (1 k × 1k)	29.7
Lowstop filter (1 k × 1k)	29.7
Non-maximum suppression (1k × 1k)	29.7
Feature extraction (1 k × 1 k)	29.7
Operations for looming targets (input size)	
Pyramid construction (3 levels)	39.2
Morphological filter (1 k × 1k)	29.7
Temporal averaging (1 k × 1k)	29.7
Dynamic programming (1 k × 1k)	29.7

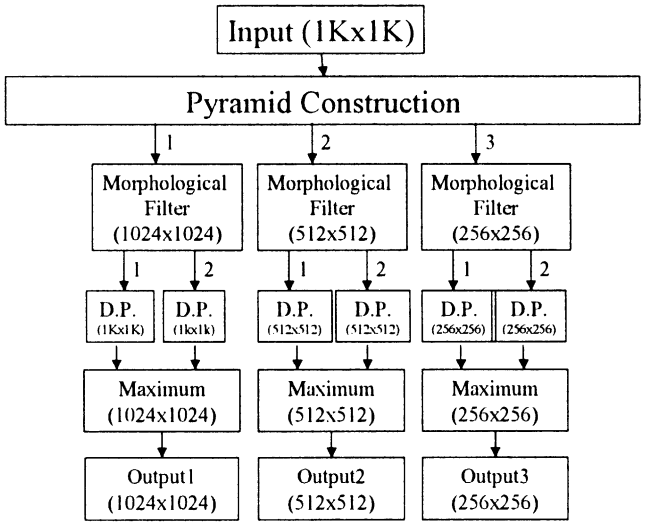

Figure 8. The algorithm for the detection of looming targets.

subsystem for translating targets (explained in previous section). The image-processing subsystem is explained below.

The image-processing subsystem for looming targets performs the basic image-processing steps to suppress clutter and extract features that could be looming targets. The algorithm uses a pyramid construction operation as a pre-processing step and performs three morphological filter operations followed by six dynamic programming operations (shown in Figure 8). The algorithms can be separated into several individual operations.

Pyramid construction. Spatial integration can be performed by forming an image pyramid to detect targets of a number of different sizes and velocities. A hierarchy of images, each with half the resolution of the previous one, was formed. A low-pass Gaussian filter was performed before down-sampling to reduce the loss of subpixel information. A three-level pyramid construction can be divided into three sequential pipes (shown in Figure 9). The first pipe smoothed and subsampled the original $1k \times 1k$ image into a 512×512 image, and copied the $1k \times 1k$ image to the destination memory surface. The second pipe smoothed and subsampled a 512×512 image into a 256×256 image, and copied the 512×512 image to the destination memory surface. Finally, the third pipe copied the 256×256 image to the destination image. These three pipes cannot be executed concurrently because they need to write into the same destination: the AM (Am1_3).

Morphological filter. A morphological filter [13] can remove large-sized features (usually clutter), while retaining small-sized features (usually targets). A difference between the original image and its morphological opening (top-hat transform) outputs small-sized positive targets – i.e. bright targets in dark background, whereas the difference between the morphological closing and the original image (bottom-hat transform) outputs negative targets, i.e. dark targets in bright background. Each of these images can be used separately to detect targets.

The flowchart of a morphological filter is shown in Figure 10(a). A branch of a morphological filter is either a dilation followed by an erosion (closing), or an erosion followed by a dilation (opening). A dilation (erosion) can be done by configuring an AU to perform a maximum (minimum) operation. Delay elements (DLY0, DLY1, DLY2, and DLY3) were included in the pipe to adjust the alignment properly (shown in Figure 10(b)).

Temporal averaging. To decrease the probabilities of false alarms and missed detections, observations can be integrated spatially or temporally. In the case of stationary targets, optimal detection can be achieved by adding (or averaging) the images in the sequence and thresholding the output. A recursive filter was used with a forgetting factor a between 0 (full forgetting) and 1 (no forgetting). The output $F(k)$ at time k is given in terms of the input $f(k)$ as

$$F(0) = 0, \quad F(k) = (1 - a)f(k) + aF(k - 1)$$

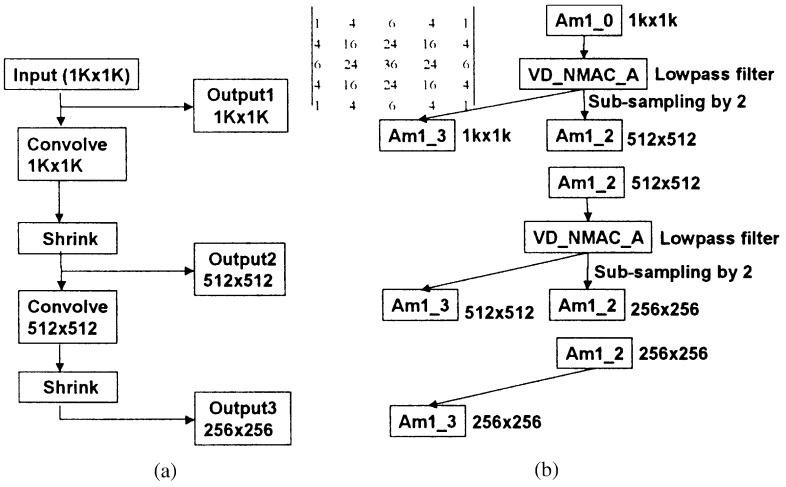

Figure 9. The implementation of pyramid construction algorithm on Datacube MaxPCI. (a) Flowchart (b) Hardware mapping.

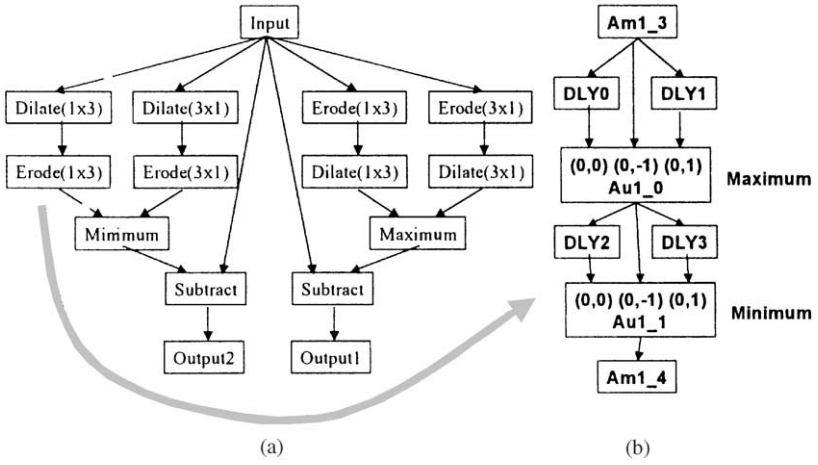

Figure 10. (a) Morphological filter. (b) The implementation of a branch of morphological filter on Datacube MaxPCI. (a) Flowchart (b) Hardware mapping.

To minimize the truncation error, a 16-bit averaging operation was used instead of an 8-bit averaging operation. An AU was configured to perform weighted 16-bit averaging. An AM unit (Am0_0) was set to store the high byte, while another AM (Am0_1) stored the low byte of the averaging output.

Dynamic programming. In the case of moving targets, the temporal averaging filter does not improve detection. A dynamic programming algorithm [3] is more effective in detecting moving targets. The algorithm is based on shifting the images before averaging them, to align the target to be detected. Since the velocity of the target could be arbitrary, the velocity space (u, v) is discretized within the range of possible target velocities. For each discrete velocity on the grid, an intermediate image F is created recursively using

$$F(x, y; u, v; 0) = 0$$
$$F(x, y; u, v; k) = f(x, y; k) + a \max_{(x', y') \in Q(x, y)} F(x' - u, y' - v; u, v; k - 1)$$

Finally, a maximum operation is performed at time K, when the result is to be reported:

$$F_m(x, y; K) = \max_{(u,v)} F(x, y; u, v; K)$$

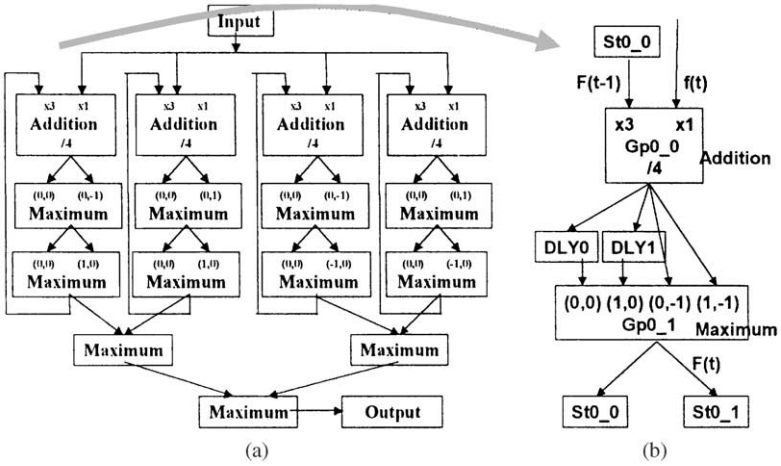

Figure 11. (a) Dynamic programming (DP) algorithm. (b) The implementation of a branch of DP algorithms on Datacube MaxPCI. (a) Flowchart (b) Hardware mapping.

Each dynamic programming consists of four branches (shown in Figure 11(a)). Each branch is a recursive temporal averaging operation with an additional dilation (maximum) operation in the loop (shown in Figure 11 (b)). A GP unit (GP0_0) was configured to perform the weighted averaging, while another GP unit (GP0_1) was configured to perform a maximum operation with four inputs. Two delay elements (DLY0, DLY1) were inserted in the proper position to achieve the correct alignments for the four inputs of the maximum operation.

Table 2 summarizes the execution throughput for the operations described in this section. The required resources are based on implementations on the Datacube MaxPCI system. The reported execution throughput is based on an input image with $1k \times 1k$ resolution. A smaller input size should result in a larger throughput.

Results of the Flight Tests

Two aircrafts were involved in flight tests at NASA Langley Research Center. A TIFS was the host aircraft, which carried the Kodak camera and Datacube computer. A Beechcraft King Air B-200 was the target aircraft. The purpose of the flight tests was to obtain images containing different maneuvers conducted by the target aircraft. For all maneuvers, the host aircraft had an altitude of 3500 ft and a speed of 159 knot.

Two classes of maneuvers were flown. In the translating maneuver (shown in Figure 3(a)), the target aircraft translated (moved) in the image sequence, which was performed with the target aircraft crossing perpendicular to the direction of motion of the host aircraft. The speed of the target aircraft was 159 knot. This maneuver was performed for different vertical and horizontal separations. Images were recorded with the target aircraft 500 ft below and 500 ft above the host aircraft at distances of about 1,2,3,4, and 5 nautical miles. Recording ended when the target aircraft left the camera's field of view.

In the looming maneuver (shown in Figure 3(b)), the target aircraft maintained a fixed position in the image surface as it flew away from the host aircraft. The target aircraft speed was 209 knot. Images were recorded, with the target aircraft ascending at 500 ft/min, descending at 500 ft/min, or maintaining a fixed altitude. Recording ended when the target aircraft was about 5 mile from the host aircraft. The images from this sequence can be played backwards to simulate the target aircraft motion that occurs in a collision.

Two flight tests to evaluate the system were conducted over several days by NASA Langley Research Center. During the first flight test in January 1999, image sequences were captured and recorded successfully at a rate of 30 frames/s. Ten real-time image sequences with translating targets, and six image sequences with looming targets were obtained, containing 86 GB (49 mins, 87,819 frames, 1 MB/frame) of digital data. The tracking algorithms were designed and fine-tuned using these image sequences. During the second flight test in September 1999, not only real-time image

capturing and recording was performed but also the translating target-tracking algorithm was executed concurrently at a rate of 15 frames/s. Algorithm output was displayed on an XVS display screen in the cockpit. Nine real-time image sequences with translating targets were obtained, containing 23 GB (26 min, 23,254 frames, 1 MB/frame) of digital data. By recording images over a multiple-day period in each flight test, a range of contrast conditions was obtained. In addition, the background of the target varied depending on its altitude. This approach provided a comprehensive set of images for testing the image-processing algorithms under different conditions.

It was observed that the system successfully detected and tracked translating objects during the flight tests. Figure 12 demonstrates a trace of the tracking algorithm applied on an image sequence with the target aircraft translating from the right to the left side of the image at a distance of 3 nautical mile. The target aircraft is located at the end of the track and occupies about 15×3 pixels in this $1\text{k} \times 1\text{k}$ image. A detection is shown by drawing a small black square around the detected position. The image was created by superimposing the detection squares through the image sequence to the last image in the sequence. The series of squares across the image shows the consistent detection of a translating target using this system. There was no false alarm in this image sequence. The distance between the host and target aircraft was 3 nautical mile in this image sequence.

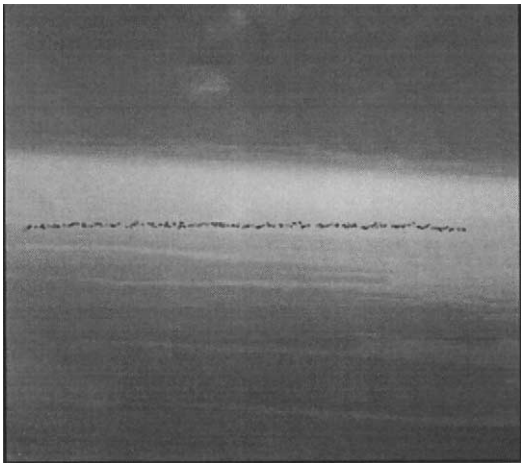

Figure 12. Tracking algorithm applied on an image sequence with the target aircraft translating from the right to the left side of the image.

The performance of the target-tracking algorithm was evaluated using the image sequences obtained from the flight tests. Table 3 summarizes the performance of the translating target-tracking algorithm for a number of distances between the host and target aircraft. The scenarios are divided into three categories: (1) scenarios with target aircraft 500 ft below the host aircraft (Scenario NO l–5); (2) scenarios with target aircraft 500 ft above the host aircraft (Scenario NO. 6–9); and (3) random traffic encounter (RTE) in that no altitude

Table 3. The performance of the translating target-tracking algorithm for a number of target distances in nautical miles (nmi). The algorithm was executed at a rate of 15 frames/s on the Datacube MaxPCI system

No.	Target Type	Target Dist.	Clutter	MD	FA
1	−	1.5		0.061	0.000
2	−	1.8		0.113	0.000
3	−	3.0	*	0.056	0.000
4	+	4.7	**	0.363	0.180
5	+	5.0	**	0.803	0.147
6	−	1.5		0.061	0.000
7	−	2.0		0.092	0.000
8	−	3.0	%	0.057	0.000
9	−	4.7	**	0.335	0.183
10	−	1.2	%%	0.159	0.063
11	−	2.4		0.059	0.000
12	−	3.0	*	0.053	0.000
13	−	3.0	%	0.089	0.000
14	−	3.0	%%	0.524	0.386
15	−	5.0	*	0.192	0.038
16	−	5.4	*%	0.643	0.000

*("%") means light ground (sky) clutter.
**("%%") means heavy ground (sky) clutter in the background through the image sequence.

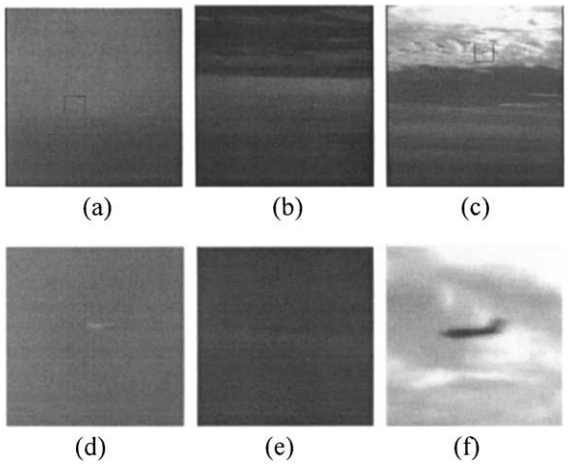

Figure 13. Three levels of background clutters and their enlarged image areas with target aircrafts: (a) No background clutter; (b) light background clutter; (c) heavy background clutter, and (d–f) enlarged image area with target in (a–c).

constraint of target aircraft was imposed on the scenarios in this category (Scenario NO 10–16). A target is classified as a positive target if it is brighter than its background, while it is classified as negative target if it is darker than its background. A target aircraft may travel from left to right, or from right to left of the host aircraft. The false alarm (FA) rate is measured as the ratio of the total number of FAs throughout the sequence to the number of image frames in the sequence. The miss detection (MD) rate is measured as the ratio of the number of frames in which the target was missed to the total number of frames. The false alarm rate depends on the amount of clutter in the images, whereas the MD rate depends on the target size and contrast, and therefore increases with the target distance in most cases. Thus, the tracking performance degrades as the amount of background clutter or the target distance increases. Figure 12 demonstrates the three levels of the background clutter and their enlarged image areas with target aircrafts. The small squares in (a–c) represent the image areas with targets that are zoomed in (d–f). The resolution of (a–c) is 1024×1024, while the resolution of (d–f) is 100×100.

In image sequence numbers 4, 5, 9, 15, and 16, the MD rates are high because the target distance is close to or longer than 5 nautical mile. In image sequence numbers 4, 5, 9, and 14, the FA rates are high because of the heavy background clutter. It is possible to get a better performance by adjusting tracking parameters individually according to the characteristics (such as clutter level and target size) of each image sequence. However, in this experiment, all tracking parameters are fixed for all sequences – not adjusted from sequence to sequence. Since FAs can be very annoying to pilots, a low FA rate was more desirable than a low MD rate. Hence, the parameters of the tracking algorithm were selected deliberately to reduce the FA rate.

In addition to background clutter and target distance, the FA rate is significantly high as the host aircraft changes direction or rotates about its own axis (such as image sequence number 14). These FAs are due to misidentification of some static background clutters as moving targets because of the relative movements. Therefore, it should be possible to reduce the system's FA rate by considering the aircraft navigation data of the host aircraft. If the aircraft navigation data for the host aircraft can be acquired as system input, then the tracking can be adjusted to compensate for the movement of host aircraft, thereby improving the target detection. If the aircraft navigation data are unavailable, then the background motion should be modeled to separate independent object motion.

Conclusions

A system was designed to capture image sequences from a digital camera with $1\,k \times 1k$ resolution at a rate of 30 frames/s, record the images into a high-speed 64 GB disk array through fiber channel, and process the images using multiple pipeline processors to perform real-time obstacle detection. The system's feasibility was demonstrated in two flight tests conducted by NASA. The first flight test focussed on image capturing and recording, while the second flight test performed image capturing, recording, and processing concurrently. All image sequences are valuable for those conducting further research on either translating or looming obstacle detection algorithms under different conditions (size, contrast, background, etc.). The system successfully detected and tracked translating objects during the flight tests.

References

1. Nishiguchi, K., Kobayashi, M. & Ichikawa, A. (1995) Small target detection from image sequences using recursive max filter. *Proceedings of SPIE*, **2561**: 153–l66.
2. Barniv, Y. (1985) Dynamic programming solution for detecting dim moving targets. *IEEE Transactions on Aerospace and Electronic Systems*, **21**(1): 144–156.

3. Arnold, J., Shaw, S. & Pasternack, H. (1993) Efficient target tracking using dynamic programming. *IEEE Transactions on Aerospace and Electronic Systems*, **29**(l): 44–56.
4. Tonissen, S. & Evans, R. (1996) Performance of dynamic programming techniques for track-before-detect. *IEEE Transactions on Aerospace and Electronic Systems*, **32**(4): 1440–1451.
5. Smith, S. (1998) ASSET-2: Real-Time Motion Segmentation and Object Tracking. *Journal of Real-Time Imaging*, **4**(1): 21–40.
6. Melton, R., Tsai, C., Alford, C. & Becker, L. (1996) A VLSI system implementation for real-time object detection. *IEEE International Symposium on Circuits and Systems*, **4**: 320–323.
7. Majumdar, A.K. (2000) Design of an ASIC for straight line detection in an image. *Proceedings of the Thirteenth International Conference on VLSI Design,* pp. 128–133.
8. Reza, A.M. & Turney, R.D. (1999) FPGA implementation of 2D wavelet transform. *Proceedings of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers,* Vol. 1, 584–588.
9. Kasturi, R., Camps, O., Coraor, L., Gandhi, T., Hartman, K. & Yang, M.T. (2000) Obstacle detection algorithms for aircraft navigation. *Technical Report*, Department of Computer Science and Engineering, The Pennsylvania State University, CSE-00-002.
10. Eastman Kodak Company. (1997) *Kodak Megaplus Camera Model ES 1.0 Opto mechanical Specification and Imaging Performance Specification*. Eastman Kodak Company.
11. Yang, M., Kasturi, R. & Sivasubramaniam, A. (2001) An automatic scheduler for real-time vision applications. *Proceedings of the International Parallel and Distributed Processing Symposium,* April 2001.
12. Datacube Inc. (1999) *PC ImageFlow Programmer's Manual*. Datacube Inc.
13. Casasent, D. & Ye, A. (1997) Detection filters and algorithm fusion for ATR. *IEEE Transactions on Image Processing*, **6**(l): 114–125.
14. Bird, J.S. & Goulding, M.M. (1992) Rate-constrained target detection. *IEEE Transactions on Aerospace and Electronic System,* **28**(2): 49l–503.
15. Burt, P. (1981) Fast filter transforms for image processing. *Computer Vision, Graphics and Image Processing*, **16**: 20–51.
16. Gandhi, T. (2000) Image sequence analysis for object detection and segmentation. *PhD thesis*, Department of Computer Science and Engineering, The Pennsylvania State University.
17. Yang, M., Gandhi, T., Kasturi, R., Coraor, L., Camps, O., & McCandless, J. Real-time obstacle detection system for high-speed civil transport supersonic aircraft. *Proceedings of the IEEE National Aerospace and Electronics Conference,* October 20, pp. 595–601.