Implementering av en detektionsalgoritm för realtidsbearbetning av IR-video

Jörgen Karlholm

FOI-R-- 1716 --SE
ISSN 1650-1942

Sensorteknik
Teknisk rapport

September 2005

FOI är en huvudsakligen uppdragsfinansierad myndighet under Försvarsdepartementet. Kärnverksamheten är forskning, metod- och teknikutveckling till nytta för försvar och säkerhet. Organisationen har cirka 1350 anställda varav ungefär 950 är forskare. Detta gör organisationen till Sveriges största forskningsinstitut. FOI ger kunderna tillgång till ledande expertis inom ett stort antal tillämpningsområden såsom säkerhetspolitiska studier och analyser inom försvar och säkerhet, bedömningen av olika typer av hot, system för ledning och hantering av kriser, skydd mot hantering av farliga ämnen, IT-säkerhet och nya sensorers möjligheter.

FOI
Totalförsvarets forskningsinstitut
Sensorteknik
Box 1165
581 11 Linköping

Tel: 013-37 80 00
Fax: 013-37 81 00

www.foi.se

Implementering av en detektionsalgoritm för realtidsbearbetning av IR-video

FOI-R--1716--SE
ISSN 1650-1942

Sensorteknik
Teknisk rapport

September 2005

Utgivare	Rapportnummer, ISRN	Klassificering
FOI - Totalförsvarets forskningsinstitut Sensorteknik Box 1165 581 11 Linköping	FOI-R--1716--SE	Teknisk rapport
	Forskningsområde 4. Ledning, informationsteknik och sensorer	
	Månad, år September 2005	**Projektnummer** E3947; E3057
	Delområde 42 Spaningssensorer	
	Delområde 2	

Författare/redaktör	Projektledare
Jörgen Karlholm	Jörgen Karlholm; Morgan Ulvklo
	Godkänd av Mattias Severin
	Uppdragsgivare/kundbeteckning FM
	Tekniskt och/eller vetenskapligt ansvarig Jörgen Karlholm

Rapportens titel

Implementering av en detektionsalgoritm för realtidsbearbetning av IR-video

Sammanfattning

Rapporten beskriver en C++-implementering i signalbehandlingshårdvara av en algoritm för detektering i realtid av markstridsfordon i IR-video från flygande plattformar som UAV:er eller styrda vapen. Algoritmen är baserad på Viola och Jones sekventiella boosting-ansats med särdrag beräknade ur en *summed area table*-representation.

Studien visar att kvalificerad realtidsdetektering i IR-video är möjlig att realisera med ett litet, billigt och strömsnålt inbyggt signalbehandlingssystem. För sekvenser i 16-bitars QVGA-format (320x240 pixlar) fås en bearbetningstakt på ca 7 Hz vid dataparallell exekvering i två MPC7457 PowerPC G4 mikroprocessorer med vardera 2MB L3-cache.

Algoritmens struktur gör det svårt att använda blockbaserad bearbetning (*L1 strip mining*), en vanlig teknik för att minimera antalet minnesaccesser vid lokala omgivningsoperationer. Bearbetningen blir därför i PowerPC G4 minnesbegränsad, men en betydande uppsnabbning kan åstadkommas genom att kritiska data allokeras i processorns L3-cache (s.k. *private memory*) istället för primärminnet.

Av kostnadsskäl är det önskvärt att kunna implementera algoritmen i enklare signalprocessorer som arbetar med 16 bitars precision. Flera databuffertar måste emellertid representeras som 32-bitars heltal, i något fall krävs 64 bitars precision. Aritmetiska operationer på dessa data måste delas upp i flera steg och tar betydligt länge tid att utföra än med 16-bitars operander.

Nyckelord

detektering, realtid, IR, spaning, övervakning, UAV, styrda vapen, målsökare

Övriga bibliografiska uppgifter	Språk Svenska
ISSN 1650-1942	**Antal sidor:** 21 s.
Distribution enligt missiv	**Pris: Enligt prislista**

Issuing organization	Report number, ISRN	Report type
FOI – Swedish Defence Research Agency Sensorteknik Box 1165 581 11 Linköping	FOI-R--1716--SE	Technical report
	Programme Areas 4. C4ISTAR	
	Month year September 2005	**Project no.** E3947; E3057
	Subcategories 42 Above water Surveillance, Target acquistion and Reconnaissance	
	Subcategories 2	

Author/s (editor/s)	Project manager
Jörgen Karlholm	Jörgen Karlholm; Morgan Ulvklo
	Approved by Mattias Severin
	Sponsoring agency FM
	Scientifically and technically responsible Jörgen Karlholm

Report title (In translation)

Implementation of detection algorithm for real-time processing of IR video

Abstract

The report describes a C++ implementation in dedicated signal processing hardware of an algorithm for real-time detection of military ground vehicles in IR video from airborne platforms, e.g., UAVs or guided weapons. The algorithm is based on Viola and Jones' sequential boosting approach with image features computed from a *summed area table* representation.

The study shows that qualified realtime detection in IR video is realizable in a small, cheap, and power-efficient signal processing system. A processing speed of 7 frames per second is achieved for 16-bit QVGA (320x240 pixel) image sequences, using two MPC7457 PowerPC G4 microprocessors, each with a 2MB L3 cache.

The structure of the algorithm makes it difficult to use block-based processing (*L1 strip mining*), a common technique for minimising I/O in local neighbourhood operations. In the PowerPC G4, the processing consequently becomes I/O bound, but a significant speed-up can be achieved by allocating critical data in the processor's L3 cache (*private memory*), instead of in the main primary memory.

For cost reasons it is desirable to implement the algorithm on 16-bit integer signal processors. Several data buffers must, however, be stored as 32-bit integers; in one case 64-bit precision is required. Arithmetic operations on these data require more processing steps, which causes a significant performance degradation compared to 16-bit data.

Keywords

detection, real-time, IR, reconnaissance, surveillance, UAV, guided missiles, seekers

Further bibliographic information	Language Swedish
ISSN 1650-1942	**Pages** 21 p.
	Price acc. to pricelist

Innehåll

1 Inledning

Autonoma system för spaning, övervakning och verkan kan förväntas få allt större betydelse i framtiden, då de ger möjlighet att operera på stora avstånd och vid störning av kommunikation. Autonom sensordataanalys har användning även när operatörer finns tillgängliga, genom att avlasta dessa från en ofta mycket monoton och tröttande arbetsuppgift.

Tränade fototolkar och sensoroperatörer är utomordentligt skickliga på att analysera bilder, givet att tid står till förfogande. I tidskritiska situationer där stora datamängder måste analyseras på kort tid kan emellertid autonom sensordataanalys förväntas väsentligt förbättra analysresultatet.

Autonom sensordataanalys kan användas för *sensoråterkopplad styrning*. Vid flyg- och UAV-spaning kan dataunderlaget förbättras genom att resultatet av dataanalysen återkopplas till själva insamlingsprocessen, exempelvis genom automatisk inzoomning av detekterade målkandidater. Ett styrt vapen kan genom realtidsanalys av sensordata planera en optimal bana mot önskad träffpunkt i målet.

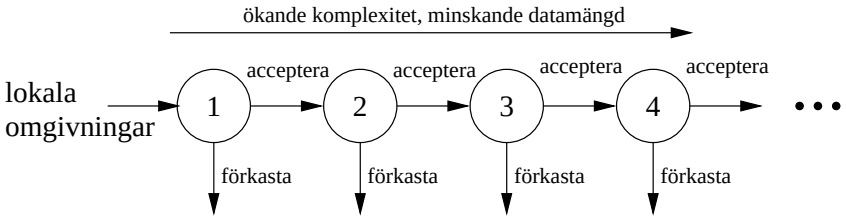
Att med hög upptäcktssannolikhet och låg falsklarmssannolikhet automatiskt detektera markstridsfordon i IR-video från en UAV kräver, p.g.a. den komplexa klottermiljön, en mycket kvalificerad klassificeringsalgoritm. Nya typer av klassificerare, exempelvis supportvektormaskiner (Schölkopf & Smola, 2002) och boosting (Hastie et al., 2001), gjorde det under 90-talet möjligt att konstruera detektionsalgoritmer med tillräcklig kapacitet. Dessa var dock så beräkningskrävande att realtidsanalys av videosekvenser var utesluten. På senare år har emellertid framsteg gjorts som möjliggjort utveckling av algoritmer för detektering i realtid. Ett av de viktigaste arbetena är Viola & Jones (2004) som presenterade en beräkningsmässigt mycket effektiv metod för ansiktsdetektering. Algoritmen som studerats i föreliggande arbete är en vidareutveckling av denna ansats.

Traditionellt har signalbehandlingsalgoritmer implementerats i digitala signalprocessorer (DSP:er), särskilt utformade för att effektivt, på stora datamängder, kunna utföra linjära filtreringar, som är den vanligaste typen av signalbehandlingsoperationer. På senare år har dock många företag istället börjat använda mikroprocessorer i sina signalbehandlingssystem. Detta gäller exempelvis samtliga större tillverkare av högpresterande inbyggda signalbehandlingssystem för militära tillämpningar: Mercury Computer Systems[1], SKY Computers[2] och CSPI[3].

Anledningen till att mikroprocessorer – som återfinns i vanliga persondatorer – blivit intressanta är främst att de tillförts s.k. multimediaenheter, utformade för realtidsbearbetning av ljud och bild i exempelvis datorspel eller vid uppspelning av strömmar över internet. Processorerna har därmed allt mer kommit att likna digitala signalprocessorer, men har den fördelen att de är betydligt enklare att programmera, vilket underlättar implementering av mer komplexa algoritmer.

Linjära filtreringar och liknande enkla sensornära algoritmer som lämpar sig för massiv parallellisering implementeras allt oftare i applikationsspecifika kretsar (ASIC) eller, i synnerhet, programmerbara logikkretsar, FPGA:er. I dessa fall exekveras inte algoritmen som ett program i mjukvara, utan i en skräddarsydd hårdvarukrets. Detta möjliggör beräkningsprestanda som är uppemot femtio gånger högre än för en signal- eller mikroprocessor. Trenden är att högpresterande signalbehandlingssystem utformas som hybridsystem där FPGA:er el-

[1] `http://www.mc.com`
[2] `http://www.skycomputers.com`
[3] `http://www.cspi.com/multicomputer`



Figur 1: Sekvens av klassificerare.

ler ASIC-kretsar utför den initiala sensornära signalbehandlingen varefter mikroprocessorer, efter reduktion av datamängden, tar över och bearbetar återstående data med mer komplexa algoritmer.

Resten av rapporten är organiserad som följer. I avsnitt 2 ges en översiktlig beskrivning av den implementerade detektionsalgoritmen. Avsnitt 3 beskriver det signalbehandlingssystem (hård- och mjukvara) i vilket algoritmen implementerats. En detaljerad beskrivning av koden ges i avsnitt 4. Avslutningsvis presenteras i avsnitt 5 slutsatser dragna av studien.

2 Detektionsalgoritmen

Den aktuella detektionsalgoritmen har i detalj beskrivits i Karlholm (2004); här följer en kort sammanfattning av relevanta delar.

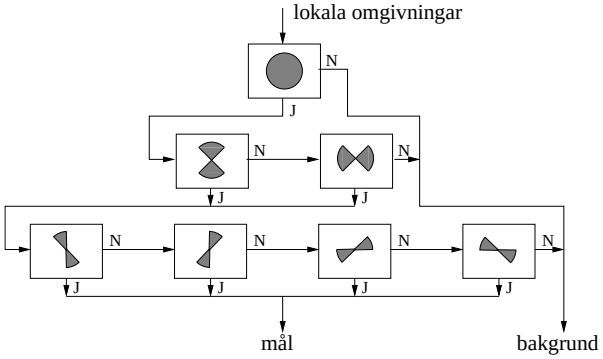
Syftet med algoritmen är att detektera markstridsfordon i IR-video från flygande plattformar, exempelvis UAV:er eller styrda vapen. Det har antagits att plattformen har så pass kvalificerade navigeringssensorer och kartmaterial att avståndet till det avbildade markplanet är känt med en osäkerhet understigande tio procent. Detta medför att målens storlek i bildplanet kan antas känd med motsvarande noggrannhet. Klassificeraren tränas att hantera återstående osäkerheter i skala samt sensorplattformens, markplanets och målets orientering.

Bearbetningen (klassificeringen) sker genom att för varje position i bilden beräkna en funktion[4] av bildelementen i den lokala omgivningen, ett rektangulärt fönster. Fönstrets storlek bestäms av det skattade avståndet till markplanet. Om funktionsvärdet överstiger en viss tröskel beslutas att omgivningen innehåller ett mål, annars att den tillhör bakgrunden.

För att uppnå beräkningsmässig effektivitet är detektorn utformad som en sekvens av allt mer kvalificerade klassificerare, som var och en tränats att eliminera en viss andel av återstående bakgrundsomgivningar i bilden, se figur 1. Om en fix andel återstående bakgrundspixlar sållas bort i varje steg minskar således antalet kvarstående pixlar exponentiellt som funktion av antalet passerade bearbetningssteg. Anledningen till att det krävs allt mer komplexa klassificerare är att en allt större andel återstående bakgrundsomgivningar är mållika.

En komplex klassificerare innehåller definitionsmässigt fler parametrar än en enkel och är svårare att träna. Ett sätt att dels underlätta träningen, dels göra bearbetningen beräkningsmässigt effektiv, är att sätta samman en komplex klassificerare genom att kombinera ett antal enklare komponenter, var och en specialiserad på en delmängd av målen. En omgivning måste då förkastas av samtliga komponenter (’experter’) för att klassificeras som bakgrund. I den

[4] Väsentligen en skattning av sannolikheten att signalen härrör från ett mål.



Figur 2: Uppdelning av bearbetningskedjan i tre steg bestående av successivt mer specialiserade experter. Var och en av komponenterna är i sin tur uppbyggda som en sekvens av klassificerare (jmf. figur 1).

aktuella implementeringen är komponenterna specialiserade till ett visst vyintervall (avseende målets orientering i markplanet), se figur 2.

I varje enskilt delsteg i detektionsalgoritmen beräknas ett skalärt funktionsvärde

$$F(\mathbf{x}) = \sum_{m=1}^{M} f_m(\mathbf{x};\gamma_m) \tag{1}$$

där $\mathbf{x}$ är en vektor bestående av bildelementen i den lokala omgivningen. $F(\cdot)$ betecknas ofta *diskriminantfunktion*, och består alltså i detta fall av en summa av basfunktioner $f_m(\cdot)$ med parametervektorer γ_m. Basfunktionerna definieras av

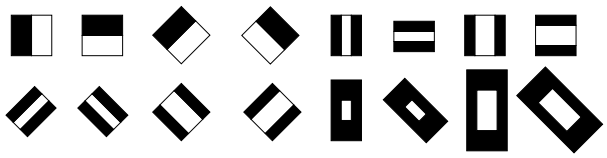
$$f_m(\mathbf{x}) = \begin{cases} c_m^R, & r_m(\mathbf{x}) \geq \sigma(\mathbf{x})\tau_m \\ c_m^L, & r_m(\mathbf{x}) < \sigma(\mathbf{x})\tau_m \end{cases} \tag{2}$$

Här är $r_m(\mathbf{x})$ utsignalen (eller dess absolutvärde) från ett linjärt filter i en viss position i den lokala omgivningen. Filtren fungerar som prober som reagerar på förekomsten av matchande lokala strukturer (särdrag) i den lokala omgivningen. De olika typer av filter som använts visas i figur 3. $\sigma(\mathbf{x})$ är standardavvikelsen för pixelvärdena i den lokala omgivningen, medan c_m^L, c_m^R samt τ_m är parametrar. Vilka filter och parametervärden som ska ingå i diskriminantfunktionen $F(\cdot)$ bestäms under träningsprocessen. Figur 4 visar några särdrag från det första steget i den implementerade detektorn. Notera att samtliga filtertyper i figur 3 kan beräknas som skillnaden mellan pixelsummorna i två rektanglar. Pixelsummorna kan, som vi ska se senare, genom en speciell transformation beräknas mycket effektivt, vilket medför att filtersvaren kan bestämmas väsentligt snabbare än genom traditionell faltning eller korrelation. Även σ kan uttryckas som en funktion av två pixelsummor i en rektangulär omgivning.

3 Bildbehandlingssystemet

3.1 Hårdvara

Detektionsalgoritmen har implementerats i ett bildbehandlingssystem från det kanadensiska företaget Matrox Electronic Systems, Ltd., en välkänd tillverkare av bildbehandlings- och gra-



Figur 3: Filtertyper använda i studien. Ett mycket stort antal olika lokala strukturer kan detekteras genom att variera filtrens position, höjd och bredd i den lokala omgivningen.

Figur 4: Fyra exempel på särdrag från första steget i detektionskedjan.

fikkort. Matrox Odyssey Xpro består av en uppsättning PCI-kort, och kan därför anslutas till en vanlig persondator. Flera kort kan kopplas samman genom snabba länkar (1 GB/s), vilket gör att PCI-bussen, som förmedlar kommunikationen med värddatorns processor, inte behöver belastas när data skickas mellan olika kort. Det aktuella systemets arkitektur visas i figur 5.

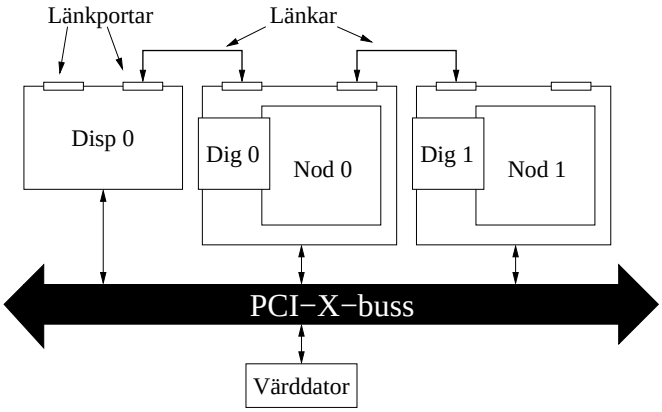
Odyssey-kortet, figur 6, styrs av en Freescale MPC7457 PowerPC G4 mikroprocessor[5] (klockfrekvens 1 GHz), som kör ett realtidsoperativsystem, MQX[6]. G4:ans minneshierarki består av separata 32 kB L1-cacheminnen för data och instruktioner, en intern 512 kB L2-cache, samt en extern L3-cache på 2 MB. Processorn kan hantera upp till 2 GB primärminne. Det tar 2-3 processorcykler att läsa data från L1-cachen, ca 10 cykler från L2-cachen och ca 40 cykler från L3-cachen. Att läsa från primärminnet kan ge en fördröjning på upp till 250 processorcykler. L3-cachen har en mycket användbar egenskap, nämligen att hela eller halva minnesutrymmet kan allokeras av användaren som ett snabbt primärminne (s.k. *private memory*), att använda för lagring av data som upprepade gånger måste läsas eller skrivas.

G4-processorn är superskalär, och kan varje klockcykel påbörja exekveringen av maximalt fyra olika instruktioner. Det finns fyra skalära 32-bitars heltalsenheter och en skalär 64-bitars flyttalsenhet. Tidsåtgången för en heltalsaddition är en klockcykel, för heltalsmultiplikation 3-4 cykler, och för flyttalsoperationer 5 cykler[7]. I de flesta fall kan varje enhet påbörja en ny beräkning varje klockcykel (s.k. *pipelining*). Liksom flera andra moderna mikroprocessorer har G4:an dessutom en särskild (multimedia-)enhet för parallellbearbetning av dataströmmar, i detta fall betecknad AltiVec. Precis som exempelvis MMX och SSE för Intel Pentium är AltiVec en SIMD-enhet (SIMD=*single instruction, multiple data*), vilket innebär att samma instruktion parallellt verkar på samtliga komponenter i en datavektor, vars längd är 128 bitar. För 8-bitars heltal motsvarar det 128/8 = 16 komponenter, för 16-bitars heltal 8 komponenter, och för 32-bitars hel- och flyttalstyper 4 komponenter. Vid exempelvis addition av två 8-bitars

[5]Data inhämtade från Freescale A (2005) och Freescale B (2003), vilka i skrivande stund kan laddas ner från Freescales webbplats `http://www.freescale.com`.

[6]`http://www.metaware.com`

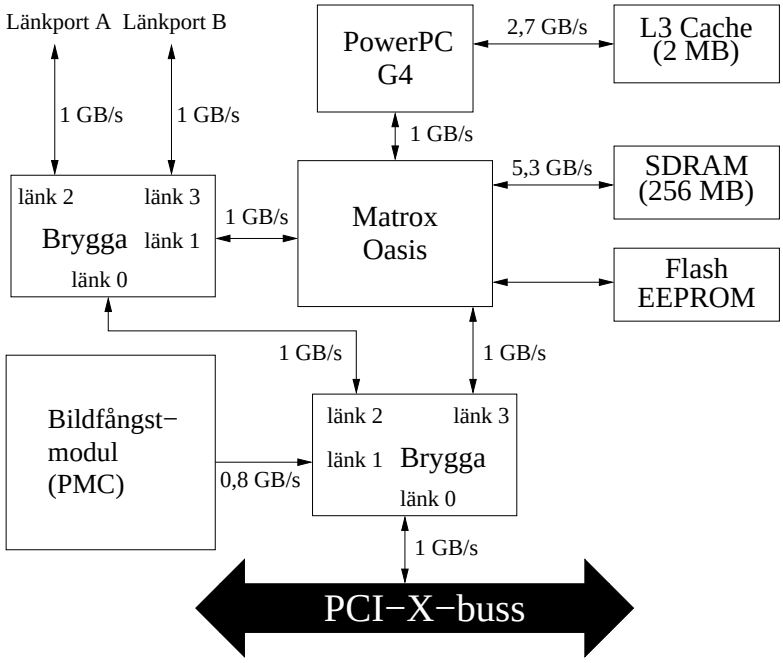
[7]För att fastställa tidsåtgången för en enskild instruktion måste även tid för hämtning och avkodning av instruktionen samt lagring av resultatet inkluderas. Totalt krävs minst 7 klockcykler i G4:an. Som jämförelse krävs i Intel Pentium 4 minst 20 klockcykler per instruktion.



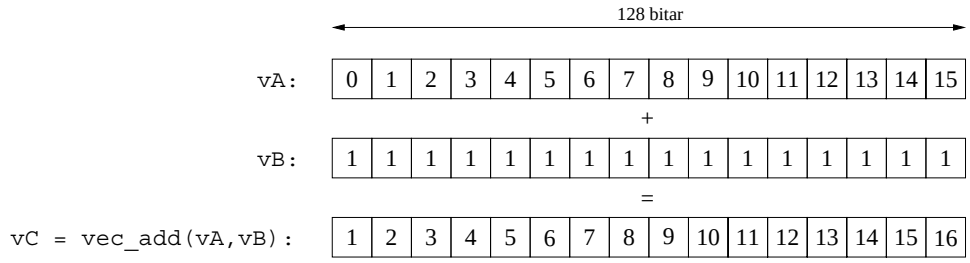
Figur 5: Arkitektur för Matrox Odyssey Pro-system bestående av två beräkningsnoder (Nod 0-1), två bildfångstenheter (dotterkort Dig 0-1), samt ett grafikkort (Disp 0). Efter Matrox A (2004).

bilder kan med AltiVec således utföras upp till 16 samtidiga skalära additioner (jmf. figur 7), vilket kan ge en stor uppsnabbning jämfört med en skalär kod. AltiVec-enheten har en uppsättning av 162 instruktioner omfattande bl.a. aritmetik, logik, permutationer och matematiska funktioner (Freescale C, 1999). I AltiVec-enheten finns fyra beräkningsenheter för vektordata: VPU för permutationer, VIU1 för enkla heltalsberäkningar, t.ex. addition, VIU2 för mer komplexa heltalsoperationer, t.ex. multiplikation, samt VFPU för flyttalsberäkningar (32 bitar). VIU1 kan utföra en operation per klockcykel, medan VIU2 och VFPU består av fyra steg, som vardera kräver en klockcykel. De senare kan dock påbörja exekveringen av en ny operation varje klockcykel (pipelining). Två olika AltiVec-instruktioner kan börja exekvera varje klockcykel, om de kan utföras i olika beräkningsenheter (t.ex. en heltals- och en flyttalsoperation). En mycket viktig delmängd AltiVec-instruktioner används för *prefetch* av dataströmmar. En prefetch-instruktion tipsar processorn om att ett visst datablock snart kommer att behövas. Detta ger processorn tid att i förväg läsa in data från primärminnet till cacheminnet, och på så sätt avsevärt minska fördröjningen vid läsning av data. Processorn kan samtidigt läsa in fyra dataströmmar från primärminnet till L1-cachen.

En central komponent på Odyssey-kortet är specialkretsen Oasis (en s.k. ASIC), som bl.a. innehåller en programmerbar beräkningsenhet, Pixel Accelerator (PA). Denna består i sin tur av 64 beräkningselement (*processing element*, PE), vilka parallellt kan utföra samma operation på olika delar av indata av 8- eller 16-bitars heltalstyp. Varje PE består av en MAC-enhet och en ALU. MAC-enheten (*multiply and accumulate*) beräknar en viktad summa av pixelvärdena i en lokal omgivning (upp till 32x32 pixlar), medan ALU:n (*aritmetic-logic unit*) utför pixelvisa operationer. Pixelacceleratorn har en klockfrekvens på 167 MHz, och såväl MAC-enheten som ALU:n kan utföra en instruktion per cykel. Pixelacceleratorn är, tack vare sin parallellbearbetningskapacitet och den stora bandbredden till primärminnet (5,3 GB/s momentant, 4.0 GB/s uthålligt) väl lämpad för grundläggande bildbehandlingsoperationer som linjära filtreringar, morfologi och pixelvisa operationer.



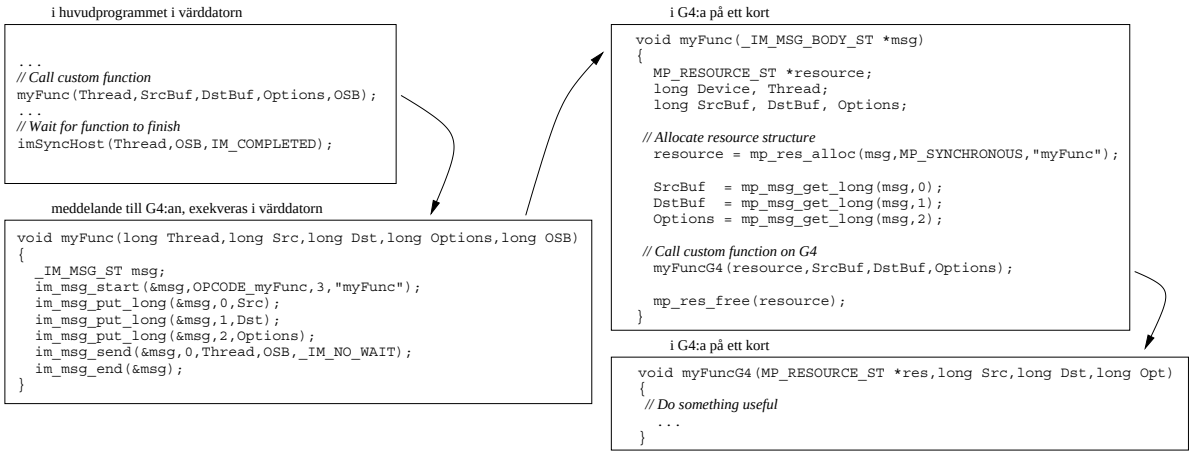
Figur 6: Odyssey-kortets arkitektur. Efter Matrox A (2004).



Figur 7: AltiVec är en SIMD-enhet som utför samma operation på samtliga element i en vektor av längd 128 bitar, vilket exempelvis motsvarar 16 stycken 8-bitars heltal.

3.2 Mjukvara

Till Matrox Odyssey Xpro finns ett stort antal funktioner som kan anropas från C/C++. På den högsta, mest abstrakta nivån finns funktionsbiblioteket MIL, *Matrox Imaging Library* (Matrox B, 2004; Matrox C, 2004). MIL är portabelt i den meningen att samma funktionsanrop fungerar med samtliga Matrox bildbehandlingskort. MIL kan även exekveras i värddatorn utan att något kort finns kopplat till systemet; koden är optimerad för MMX och SSE. För att programmera med MIL krävs inga detaljerade kunskaper om systemarkitekturen, utan biblioteket bestämmer själv var funktionen ska exekveras (i värddatorn eller i en viss enhet på ett visst kort), samt vilken väg data ska överföras mellan olika enheter. Flera processer som anropar MIL kan köras samtidigt (s.k. multi-tasking). Dessutom har MIL stöd för multipla trådar. Trådar är separata exekveringsköer i samma process. Trådarna delar på processens globala data och addresser, och kan kommunicera och utbyta lokala data med varandra. Trådning är lämpligt för att parallellisera uppgifter som är i stort sett oberoende av varandra, men kräver tillgång till samma



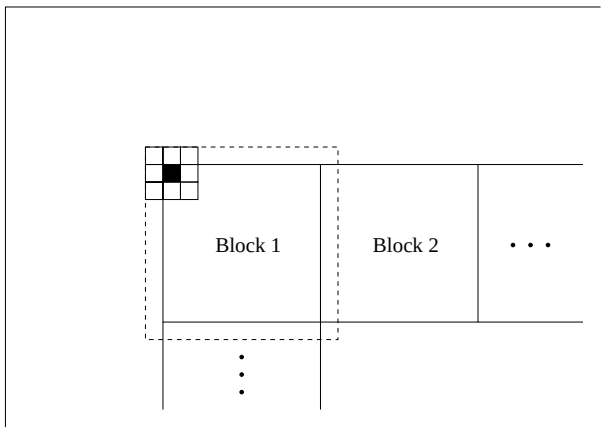
Figur 8: Ett funktionsanrop i värddatorn paketeras och skickas till G4:an som ett meddelande. I G4:an avkodas meddelandet av angiven tråd och 'nyttofunktionen' anropas.

datauppsättning (t.ex. en bild) och utbyte av viss information för exempelvis synkronisering.

Ett steg närmare hårdvaran finns funktionsbiblioteket ONL, *Odyssey Native Library*, (Matrox E, 2004; Matrox F, 2004). ONL fungerar endast med Matrox Odyssey-kort och ger en bättre kontroll över hur och var i systemet operationer utförs, i synnerhet för system med flera noder. Vid anrop av en ONL-funktion specificeras alltid vilket enhet (värddatorn eller ett specifikt kort) som ska exekvera koden. Detta sker indirekt genom att funktionsanropet placeras i en exekveringskö som i förväg allokerats till en viss enhet. Funktionerna i varje kö exekveras seriellt (dvs i turordning). De flesta ONL-funktioner är asynkrona, vilket betyder att kontrollen omedelbart returneras till värddatorn så fort kommandot skickats till den aktuella exekveringskön (funktionen blockerar inte huvudprogrammet). Funktionerna returnerar således inga meningsfulla data, utan information om att exekveringen avslutats på korrekt sätt får inhämtas på annat sätt, exempelvis genom att specificera en felhanteringsfunktion som automatiskt anropas om fel uppstår. Exekvering av funktioner i olika trådar synkroniseras genom att läsa ett speciellt minnesblock (*Operation Status Block*, OSB) där information om en funktions aktuella tillstånd skrivs. På detta sätt kan en funktion vänta tills en annan uppnått ett visst tillstånd, t.ex. startat eller avslutat. Synkronisering kan exempelvis användas för att i flera trådar parallellt bearbeta olika delar av en bild; trådarna väntar då på varandra innan nästa bild läses in.

Exekveringen i bildbehandlingssystemet av såväl MIL- som ONL-funktioner sker genom att ett meddelande sätts samman i värddatorn och skickas till aktuell G4-processor, där meddelandet tolkas och den faktiska funktionen anropas med aktuella parametrar. Figur 8 illustrerar hur ett funktionsanrop i värddatorn länkas vidare till ett kort. Den grundläggande datatypen i såväl MIL som ONL betecknas *buffert*. En buffert kan innehålla en bild, en vektor eller styrparametrar till en funktion. Endast data som paketerats i en buffert kan skickas mellan olika noder. Buffertar kan från värddatorn emellertid allokeras i primärminnet på ett kort, så funktionsanropet i figur 8 behöver inte nödvändigtvis medföra att det sker en dataöverföring mellan värddatorn och kortet.

Som framgår av figur 8 kan man själv skriva C/C++-kod som fungerar som MIL- och ONL-funktioner. Detta kräver programpaketet *Odyssey Developer's Toolkit* (DTK), (Matrox



Figur 9: Blockbearbetning (*strip mining*). För att beräkna faltningssvaret från en 3×3-kärna krävs läsning av nio pixelvärden. En naiv implementering av faltning av en bild med M rader och N kolumner kräver således i värsta fall läsning av $9MN$ pixlar från primärminnet. Vid blockbearbetning delas bilden in i icke-överlappande block. Ett block i taget, plus en omgivande ram av grannpixlar (streckat), läses för bearbetning in från primärminnet till cacheminnet. Beräkning av filtersvaren för pixlarna i blocket kräver nu ingen ytterligare läsning från primärminnet. Endast pixlar i randen av varje block läses in till cachen mer än en gång. Om blocken är stora i förhållande till filterstorleken kan detta ge en stor uppsnabbning.

D, 2004; Matrox G, 2003). Med DTK kan man skriva programkod för G4:an (inklusive AltiVec) och där göra anrop till MIL- och ONL-funktioner på samma sätt som i värddatorn. Detta medför att värddatorn fullständigt kan avlastas och all exekvering istället utföras i bildbehandlingssystemet. Ytterligare fördelar med DTK är att man kan programmera pixelacceleratorn (i ett assemblerliknande språk), samt utnyttja en uppsättning lågnivåfunktioner vid programmering av G4:an. Dessa funktioner används vid blockbaserad bearbetning, där en bild delas upp i rektangulära block som, ett i taget, läses in i L1-cachen och sedan bearbetas. Tekniken, som ofta kallas *strip mining* (dagbrytning), illustreras i figur 9.

4 Implementeringen

4.1 Parallellisering mellan korten

Den aktuella applikationen ska i realtid bearbeta 16-bitars IR-videodata som läses in via en bildfångstmodul ansluten till ett av Odysseykorten. Det är naturligt att fördela arbetet mellan de båda korten på så sätt att vardera kortet bearbetar halva bilden, s.k. *dataparallell* bearbetning, vilket i detta fall är att föredra framför annan uppdelning, exempelvis *pipelining*, där ett kort gör en första del av bearbetningen av en bild och sedan skickar delresultatet till nästa för avslutning, eller *round-robin*, där vardera kortet bearbetar varannan bild. Det finns, i den aktuella tillämpningen, ingen uppenbar fördel med pipelining i förhållande till dataparallell bearbetning, men däremot en tydlig nackdel, nämligen att delresultatet måste skickas mellan korten, vilket kan medföra en signifikant tidsfördröjning. Round-robin ger en något enklare kod, men har den nackdelen att, även om bearbetningstakten (bilder per sekund) kan hållas

lika hög som vid dataparallell bearbetning, tar det dubbelt så lång tid att bearbeta en enskild bild, vilket kan ge en tidsfördröjning som är problematisk i en realtidstillämpning.

4.2 Kodstruktur

4.2.1 Översikt

Koden är skriven i ANSI C++ med tillägg av AltiVec-instruktioner (Freescale C, 1999) samt funktionsanrop till Odyssey Native Library (Matrox E, 2004; Matrox F, 2004) och Odyssey Developer's Toolkit (Matrox G, 2003).

Explicit hantering av minnesutrymme i L1- och L3-cachen – vilket möjliggjorde en påtaglig prestandaförbättring – kräver normalt assemblerprogrammering, men i det aktuella fallet sköttes detta via anrop till ONL- och DTK-funktioner.

En pseudokod på hög nivå visas i figur 10. När båda korten utnyttjas körs denna kod i båda G4:orna. Nedan följer en mer detaljerad beskrivning av de olika delarna av koden. I texten anges beräkningstider för olika delmoment. Det bör noteras att tidsåtgången för bearbetningen av en bild varierar beroende på bildinnehållet. Värdena som anges är snittvärden över en testsekvens.

4.2.2 SAT-transformen

Utgående från en bildtransform kallad *summed area table* (SAT) (Lienhart & Maydt, 2002; Viola & Jones, 2004) kan de pixelsummor som ingår i filtersvar av den typ som visas i figur 3 beräknas mycket effektivt. I SAT-transformen ersätts pixelvärdet i varje position med summan av pixelvärdena i en rektangel vars nedre högra hörn är den aktuella positionen. I den roterade SAT-transformen (RSAT) ersätts pixelvärdet med summan av pixelvärdena i en 45 grader roterad rektangel med nedersta hörnet i aktuell position. Pixelsumman i en godtycklig stående eller roterad rektangel kan sedan beräknas från fyra tabellvärden, se figur 11.

SAT-transformen kan beräknas genom att först bilda kumulativa pixelsummor över alla kolumner, och sedan över alla rader. Mer effektivt är dock att använda rekursionen

$$SAT(x,y) = SAT(x,y-1) + SAT(x-1,y) - SAT(x-1,y-1) + I(x,y)$$

med randvärden $SAT(x,-1) = SAT(-1,y) = SAT(-1,-1) = 0$. Den roterade transformen beräknas med rekursionen

$$RSAT(x,y) = RSAT(x-1,y-1) + RSAT(x+1,y-1) - RSAT(x,y-2) + I(x,y) + I(x,y-1)$$

med randvärdena

$$RSAT(-1,y) = RSAT(x,-1) = RSAT(x,-2) = RSAT(-1,-1) = RSAT(-1,-2) = 0$$

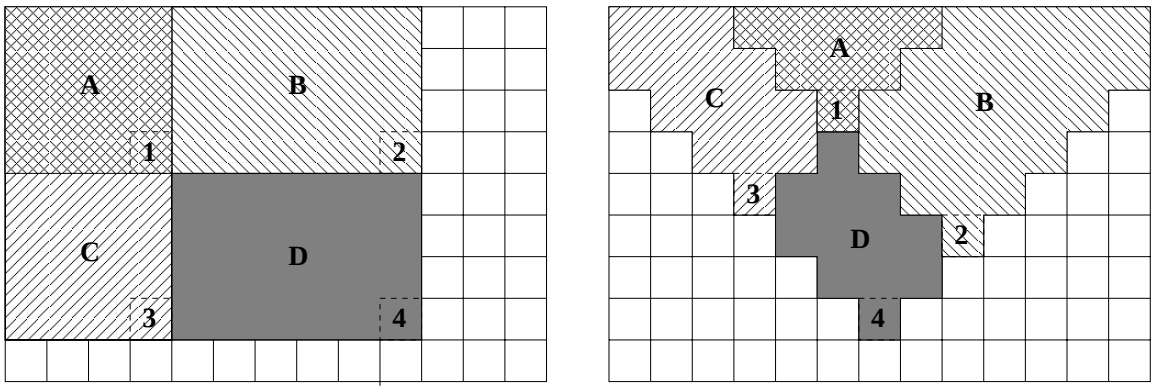
Rekursioner som dessa lämpar sig inte för AltiVec-implementering. Parallellism kan likväl uppnås (minns att G4:an har fyra oberoende skalära heltalsenheter) genom s.k. *loop unrolling* där, med lite trixande, fyra pixlar kan bearbetas i varje iteration. För en 16-bitars bild med 240×320 pixlar kan SAT-transformen beräknas på 0.7 ms och RSAT-transformen på 1.0 ms. Resultaten lagras som 32-bitars teckenlösa heltal. För att effektivt beräkna standardavvikelsen för pixelvärdena i det lokala fönstret kring varje pixel krävs också SAT-transformen av den pixelvis kvadrerade bilden. Detta resultat måste lagras som 64-bitars flyttal; beräkningen tar 2.4 ms.

```
<allokera resurser>
while (true)
{
  <synkronisera noderna, läs in bild>
  <beräkna SAT-transform, lagra i L3-cachen>
  <beräkna RSAT-transform, lagra i L3-cachen>
  <beräkna SAT-transform för kvadrerad bild>
  <bestäm skala för varje pixelomgivning, lagra i L3-cachen>
  <beräkna standardavvikelse för varje pixelomgivning, lagra i L3-cachen>
  <initiera Indikeringsobjekt indikeringar>
  for nivå=1:NIVÅANTAL
  {
    <initiera tomt Indikeringsobjekt indikeringsumma>
    for kedja=1:KEDJEANTAL(nivå)
    {
      tmp_indikeringar=indikeringar
      for steg=1:STEGANTAL(kedja)
      {
        <evaluera diskriminantfunktionen F(kedja,steg) för tmp_indikeringar>
        <eliminera tmp_indikeringar för vilka F(kedja,steg)<0 >
      }
      indikeringssumma = indikeringssumma ∪ tmp_indikeringar
    }
    indikeringar := indikeringssumma
  }
  <lagra kvarstående indikeringar>
}
<frigör allokerade resurser>
```

Figur 10: Pseudokod för implementering av detektionsalgoritmen.

4.2.3 Lokal skala

Som tidigare nämnts beräknas till varje bild en avståndsbuffert, utgående från navigationsdata och en terrängmodell. Från avståndsbufferten kan den lokala skalan relativt den referens för vilken klassificeraren är tränad bestämmas för varje pixel i bilden. I referensskalan består den lokala omgivningen av 26×26 pixlar. För att applicera detektionsalgoritmen i en viss pixelomgivning måste position och storlek för de filter som ingår anpassas till aktuell skala. Varje filter lagras som en lista av heltal representerande position och storlek i referensskalan för ingående rektanglar. För att snabba upp omvandlingen till den aktuella skalan implementeras detta som en tabellslagning. I förväg beräknas en tabell av heltal $ReScale(val, scale)$ för $val = 1, 2, \ldots, WINSIZE$, och $scale = (1.x)^n,\ n = 0, 1, 2, \ldots, NMAX$. Skalfaktorn $1.x$ kan anpassas till den skaltolerans som byggts in i klassificeraren; i det aktuella fallet har 1.1 använts. För varje bild bestäms sedan i varje pixel den aktuella skalexponent n som ger ett skalvärde närmast det ur navigationsdata beräknade värdet. För en 16-bitars bild med 240×320 pixlar tar



(a) (b)

Figur 11: Beräkning av pixelsumma i ett rektangulärt fönster från *summed area table* (SAT). (a) Ordinär SAT-transform. (b) Roterad SAT (RSAT). I båda fallen är (R)SAT(1) i position 1 pixelsumman i området A i ursprungsbilden, (R)SAT(2) pixelsumman i A∪B, (R)SAT(3) summan i A∪C, och (R)SAT(4) summan i A∪B∪C∪D. Således är summan i området D (R)SAT(1)+(R)SAT(4)-(R)SAT(2)-(R)SAT(3).

denna beräkning 3.8 ms; då ingår också att fastställa för vilka pixlar det lokala fönstret ryms inom bildens kanter. Koden använder blockbearbetning och AltiVec-instruktioner, vilket ger en signifikant uppsnabbning jämfört med en naiv skalär kod (slinga över rader och kolumner), vars exekvering tar 24 ms.

4.2.4 Standardavvikelse

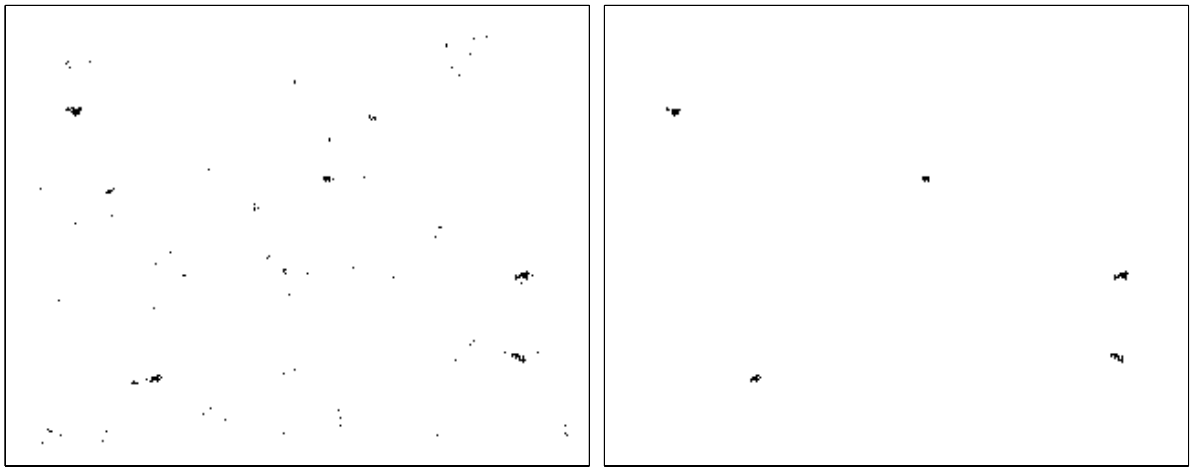
För att beräkna standardavvikelsen för pixelvärdena $I_1, \ldots, I_N$ i det lokala fönstret kring varje pixel utnyttjas den välkända formeln

$$\sigma^2 = \frac{1}{N-1}[\sum_{i=1}^{N} I_i^2 - \frac{1}{N}(\sum_{i=1}^{N} I_i)^2]$$

vilket visar att standardavvikelsen σ är en funktion av pixelsumman och summan av kvadrerade pixlar. Beräkningen av σ^2 utförs i en slinga över rader och kolumner, där i varje pixel koordinaterna för det lokala fönstrets hörn bestäms genom slagning i skalningstabellen *ReScale*$(\cdot,\cdot)$, så som beskrivits ovan. Givet hörnpunkterna kan pixelsummorna bestämmas ur SAT-transformen.

G4-processorn beräknar skalära kvadratrötter genom en märkligt långsam mjukvarualgoritm. Det finns dock en AltiVec-instruktion för beräkning av kvadratroten ur inversen av ett tal. Genom att multiplicera resultatet med originalvärdet fås en skattning av kvadratroten. Detta ger en kolossal uppsnabbning jämfört med den skalära operationen. Om kvadratrotsfunktionen anropas för varje pixel i slingan blir tidsåtgången för beräkningen av σ för en 16-bitars bild med 240×320 pixlar 51 ms. Om istället σ^2 beräknas i slingan och kvadratroten genom en efterföljande blockbearbetning med AltiVec-instruktioner blir tidsåtgången 12.3 ms, varav den senare fasen kräver 0.3 ms.

(a)



(b) (c)

Figur 12: Filtrering av indikeringar. (a) Testbild. (b) Slutresultat utan filtering. (c) Med filtering. Notera falsklarmet mitt i bilden.

4.2.5 Indikeringar

Det är viktigt att i detektionskedjan utnyttja att de flesta lokala omgivningar snabbt sållas bort. För varje aktuell skala lagras därför koordinaterna för återstående pixlar i en lista. Vidare utnyttjas det faktum att indikeringarna för verkliga mål tenderar att bilda ansamlingar (kluster), medan de flesta falsklarm endast består av några enstaka isolerade pixlar. Efter varje steg i detektionskedjan avlägsnas därför indikeringar som inte tillhör tillräckligt stora kluster, vilket ger en stor tidsbesparing i den fortsatta bearbetningen, se figur 12. För att underlätta denna filtrering lagras indikeringarna också i en tvådimensionell binär matris.

Hanteringen av indikeringar (jmf. pseudokoden i figur 10), inklusive trösklingen av diskriminantfunktionerna (1), tar sammanlagt ca 25 ms för en 16-bitars bild med 240×320 pixlar.

4.2.6 Evaluering av diskriminantfunktionen

Den helt dominerande delen av beräkningstiden åtgår vid evalueringen av diskriminantfunktionerna (1) i detektionskedjan. Var och en av 'expert'-modulerna i figur 2 består av en kedja av klassificerare enligt figur 1. Den första modulen består exempelvis av fyra steg med 16, 29, 40 respektive 51 basfunktioner (filter).

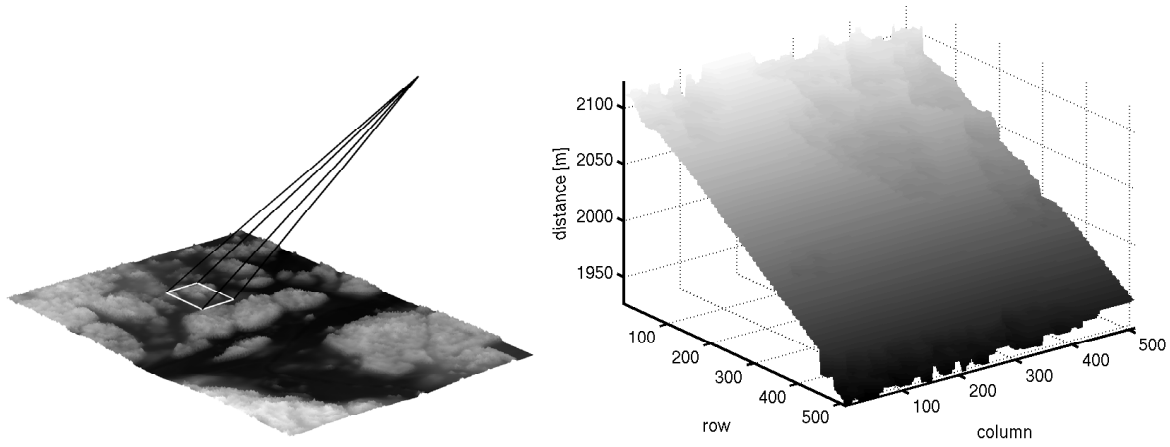
För att evaluera en basfunktion i en pixel görs först en viktad summering av ett antal SAT-värden i den lokala omgivningen (jmf. figur 11), följt av punktvisa operationer på resultatet (se (2)). Detta är snarlikt den typ av operationer för vilken pixelacceleratorn är utformad (se avsnitt 3.1). Skillnaden är att MAC-enheten i pixelacceleratorn implementerar translationsinvarianta operationer, t.ex. faltning, medan viktkoefficienterna i det aktuella fallet varierar över bilden, som funktion av den lokala skalan. Även om skalan vore konstant skulle pixelacceleratorn inte kunna användas för att beräkna filtersvaren, eftersom SAT-transformerna måste lagras som 32-bitars heltal, och MAC-enheten endast kan hantera 8- och 16-bitars heltal.

Ett alternativ till pixelacceleratorn är blockbearbetning i G4:an (jmf. figur 9). Även här stöter man dock på problem, nu i form av den begränsade storleken på L1-cachen, 32 kB[8], vilket motsvarar 8192 stycken 32-bitarstal. Detta är – eftersom de lokala pixelomgivningarna består av minst 26×26 pixlar – ett alldeles för litet antal för att åstadkomma någon uppsnabbning jämfört med en rättfram slinga över rader och kolumner. Resultatet är att man tvingas läsa in varje SAT-värde 6-8 gånger från lägre nivåer i minneshierarkin. Det är dock trots allt möjligt att signifikant snabba upp minnesaccesserna, nämligen genom att allokera SAT-transformerna i L3-cachen (*private memory*) istället för primärminnet. Som framgår av figur 6 är bandbredden betydligt större till L3-cachen än till primärminnet. Än viktigare är att fördröjningen vid läsning är avsevärt lägre.

Vid beräkningen av filtersvar kan man utnyttja att skalan vid spaning över någorlunda plan terräng tenderar att vara åtminstone styckvis konstant längs en rad i bilden, mätt med den tolerans (skalfaktor, se avsnitt 4.2.3) som byggts in under träning av klassificeraren. Längs ett sådant segment är successiva accesspunkter i SAT-transformen belägna på samma rad. Fördröjningen vid inläsning kan då minskas genom kontinuerlig prefetch av data längs aktuella rader. Om skalexponenten (se avsnitt 4.2.3) är konstant längs en rad kan särdragsumman effektivt beräknas genom faltning. Prefetchen är emellertid så effektiv att om mer än ca 40% av pixelomgivningarna längs en rad har samma skala lönar det sig att falta raden i den skalan och därefter räkna om resultatet för de pixlar som har en annan skala. (I själva verket varierar den lokala skalan vid UAV-spaning ofta så litet att samma skalexponent kan användas för hela bilden, jmf. figur 13). Med denna metod kan en särdragssumma beräknas på ca 1 ms för för en 16-bitars bild med 240×320 pixlar; den begränsande faktorn är här bandbredden till L3-cachen. De punktvisa ickelinjära operationer som ingår i beräkningen av filtersvaret (se (2)) kan på rader med många pixlar i samma skala beräknas effektivt med hjälp av blockbearbetning och AltiVec-instruktioner; tidsåtgången för detta delsteg blir då ca 1.25 ms. Sammantaget tar evalueringen av diskriminantfunktionen (1) i första detektorsteget ca $16\times(1.0+1.25)=36$ ms. Tillsammans med efterföljande tröskling och filtrering av indikationerna ger detta en total tidsåtgång på 39 ms.

Faltningsmetoden är bara av intresse i det första steget av detektorn, eftersom antalet återstående bakgrundspixlar snabbt reduceras i beräkningsskedjan, samtidigt som antalet basfunktioner i varje steg ökar. I de senare stegen är vinsten med prefetch, vektorisering, loop-

[8] I själva verket är typiskt endast halva detta utrymme tillgängligt för blockbearbetning.



(a) (b)

Figur 13: Lokal skala. Med ett synfält på 5° och en fokalplansmatris med 512×512 pixlar motsvarar detektorns referensskala ett målavstånd på 2000 m. Även ganska stora höjdvariationer i den avbildade terrängen resulterar då i endast små variationer i lokal skala i bilden.

unrolling och liknande knep minimal, utan den största uppsnabbningen fås genom att allokera data i L3-cachen istället för primärminnet, vilket avsevärt minskar accesstiderna. För samma bildformat som ovan fås en tidsåtgång för steg 2-4 i den första detektorkedjan på ca 54, 37 respektive 28 ms. De båda kedjorna på nästa nivå (se figur 2) tar vardera ca 33 ms, medan de fyra kedjorna på tredje nivån vardera tar ca 4 ms.

En sammanställning av av beräkningstiderna för olika delmoment i algoritmen ges i tabell 1. De totala tidsåtgången vid bearbetning av en 16-bitars bild med 240×320 pixlar i ett Odyssey-kort blir ca 270 ms. Om bearbetningen parallelliseras så att två kort vardera bearbetar halva bilden fås en tidsåtgång på ca 150 ms, dvs en bearbetningstakt av knappt 7 bilder per sekund.

5 Slutsatser

Studien visar att kvalificerad realtidsdetektering av markstridsfordon i IR-video från flygande plattformar är möjlig att realisera med ett litet, billigt och strömsnålt inbyggt signalbehandlingssystem. Detta ger stora möjligheter att höja förmågan hos spanings- och verkanssystem som UAV:er och styrda vapen. Om algoritmen kombineras med en mekanism för att med sensorn automatiskt söka av den överflugna terrängen (se Skoglar et al. (2005)) kan sensoroperatören fokusera på att verifiera indikeringar genererade av detektionsalgoritmen. Denna typ av avlastning genom automatisering av lågnivåoperationer kan höja tempot i insamlingen och analysen av sensordata och därmed öka förmågan att verka mot tidskritiska mål och vid hot mot plattformen.

Räknat i antal operationer per pixel är den implementerade detektionsalgoritmen mycket effektiv. På bara några enstaka millisekunder beräknas de bildtransformer från vilka en basfunktion i en pixelomgivning kan evalueras med en handfull operationer. De mest kritiska

operation	*tid [ms]*
Beräkna SAT-transform, lagra i L3-cachen	0.7
Beräkna RSAT-transform, lagra i L3-cachen	1.0
Beräkna SAT-transform för kvadrerad bild	2.4
Bestäm skala för varje pixelomgivning, lagra i L3-cachen	3.8
Beräkna standardavvikelse för varje pixelomgivning, lagra i L3-cachen	12
Nivå 1, lager 1 (16 särdrag)	39
Nivå 1, lager 2 (29 särdrag)	54
Nivå 1, lager 3 (40 särdrag)	37
Nivå 1, lager 4 (51 särdrag)	28
Nivå 2, kedja 1–2	$33+33$
Nivå 3, kedja 1–4	$4+4+4+4$

Tabell 1: Tidsåtgång för olika delmoment vid bearbetning av en 16-bitars QVGA-bild i en processor (jmf. med figurerna 2 och 10).

delarna av koden domineras i själva verket av läsning och skrivning till minnet. Detta beror på att accessmönstret är sådant att det är svårt att utnyttja blockbearbetning, vilket annars skulle minska antalet gånger varje pixel måste laddas in till processorns L1-cache.

Den låga bandbredden till primärminnet är en välkänd svaghet i arkitekturen för PowerPC G4. I den aktuella implementeringen har detta kunnat kringgås genom att allokera kritiska databuffertar som *private memory* i L3-cachens snabba SRAM-minne.

Ett problem med algoritmen är att SAT-transformen kräver att flera databuffertar måste allokeras som 32-bitars datatyper, i något fall krävs 64-bitars precision. Detta medför att Odyssey-kortets Oasis-krets – en massivt parallell SIMD-enhet för lokala omgivningsoperationer – inte kan användas, då den endast kan hantera 8- och 16-bitars indata. Av kostnadsskäl är det önskvärt att kunna implementera algoritmen i enklare digitala signalprocessorer som arbetar med 16 bitars precision. Aritmetiska operationer på 32- eller 64-bitars data måste i dessa processorer delas upp i flera steg, och tar därför betydligt längre tid att utföra än med 16-bitars operander. För att direkt hantera data med 32 eller 64 bitars precision krävs mer komplexa och dyrare processorer, exempelvis mikroprocessorer som PowerPC och Intel Pentium, eller signalprocessorer som TigerSHARC från Analog Devices[9] och C64/C67 från Texas Instruments[10]. Även avancerade FPGA:er[11] kan vara aktuella.

Bearbetningstiden för varje steg i detektionskedjan är direkt proportionell mot antalet termer i diskriminantfunktionen (1). I Karlholm (2004) visades att valet av basfunktionsuppsättning har stor påverkan på detektionsprestanda; ju större diskrimineringsförmåga basfunktionerna har, desto färre termer krävs i diskriminantfunktionen för att uppnå fastställda prestandakrav. Nyligen har det visats att en ökad diskrimineringsförmåga kan uppnås genom att applicera SAT-filtren på kantbilder (Levi & Weiss, 2004). Efter att den lokala orienteringen bestämts i varje punkt i bilden kan, för en rektangelformad region, summan av antalet pixelomgivningar med orientering inom ett visst vinkelintervall beräknas med SAT-transformen. Genom att kombinera sådana rektanglar kan skillnader i dominant orientering mellan olika regioner effektivt detekteras. I princip kan metoden utvidgas till andra texturdeskriptorer.

[9] `http://www.analog.com`

[10] `http://www.ti.com`

[11] Se exempelvis `http://www.nallatech.com`

Referenser

Freescale A (2005). *MPC7457 RISC Microprocessor Hardware Specifications*. Freescale Semiconductor, Inc. Technical Data MPC7457EC.

Freescale B (2003). *Cache Latencies of the PowerPC MPC7451*. Freescale Semiconductor, Inc. Application Note AN2180/D.

Freescale C (1999). *AltiVec Technology Programming Interface Manual*. Freescale Semiconductor, Inc. Manual ALTIVECPIM/D.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning; data mining, inference, and prediction*. Springer-Verlag, New York. ISBN 0-387-95284-5.

Karlholm, J. (2004). Implementation and evaluation of a method for detection of ground targets in aerial EO/IR imagery. Scientific report FOI-R–1267–SE, Swedish Defence Research Agency.

Levi, K. & Weiss, Y. (2004). Learning object detection from a small number of examples: The importance of good features. I *Proceedings of the International Conference on Computer Vision and Pattern Recognition*.

Lienhart, R. & Maydt, J. (2002). An extended set of Haar-like features for rapid object detection. I *Proceedings of the IEEE International Conference on Image Processing*, volume 1, pages 900–903.

Matrox A (2004). *Matrox Odyssey Installation and Hardware Reference*. Matrox Electronic Systems, Ltd. Manual no. 10851-101-0100.

Matrox B (2004). *Matrox Imaging Library version 8 Command Reference*. Matrox Electronic Systems, Ltd.

Matrox C (2004). *Matrox Imaging Library version 8 User Guide*. Matrox Electronic Systems, Ltd.

Matrox D (2004). *Matrox Imaging Library function Developer's Toolkit for Matrox Odyssey. Developer's Guide*. Matrox Electronic Systems, Ltd.

Matrox E (2004). *Matrox Odyssey Native Library Command Reference*. Matrox Electronic Systems, Ltd. Manual no. 10854-701-0200.

Matrox F (2004). *Matrox Odyssey Native Library User Guide*. Matrox Electronic Systems, Ltd. Manual no. 10852-301-0200.

Matrox G (2003). *Matrox Odyssey Native Library Developer's Toolkit Programming Guide*. Matrox Electronic Systems, Ltd. Manual no. 10853-801-0100.

Schölkopf, B. & Smola, A. J. (2002). *Learning with Kernels – Support Vector Machines, Regularization, Optimization and Beyond*. The MIT Press. ISBN 0-262-19475-9.

Skoglar, P., Björström, R., Nygårds, J., & Ulvklo, M. (2005). Information-theoretic approach for concurrent path and sensor planning for a UAV with EO/IR sensors. Scientific Report FOI-R–1685–SE, Swedish Defence Research Agency. ISSN-1650-1942.

Viola, P. & Jones, M. (2004). Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154.