Ontwerp van een patroonherkenningsmodule voor een snel naadvolgsysteem

de Deckere, W.J.M.A.

Award date:
1990

TECHNISCHE UNIVERSITEIT EINDHOVEN
FACULTEIT DER ELEKTROTECHNIEK
Vakgroep Meten en Regelen

Ontwerp van een patroonherkenningsmodule voor een snel naadvolgsysteem

W.J.M.A. de Deckere

Afstudeerraport van de doctoraal studie Elektrotechniek
Januari 1989 t/m december 1989

Mentoren: Prof. Ir. F.J. Kylstra
en Ir. R.G. van Vliet.

Deckere, W.J.M.A. de, "Ontwerp van een patroonherkenningsmodule voor een snel naadvolgsysteem"
afstudeerverslag, vakgroep ER, faculteit der Elektrotechiek,
Technische Universiteit Eindhoven, jan 1990.

In het kader van het project "Snel naadzoeken en naadvolgen" wordt op de vakgroep Meten en Regelen gewerkt aan de realisatie van een robotsysteem dat in staat is om de naden in een werkstuk te volgen met een snelheid van 0.5 m/s en een nauwkeurigheid van 1 mm. Om aan deze snelheids- en nauwkeurigheidseis te kunnen voldoen maakt het besturingssysteem gebruik van sensorinformatie om de robot langs de naden te bewegen. De geometrische vorm van de naden wordt zichtbaar gemaakt door er een streepjespatroon op te projecteren, dat bestaat uit een aantal parallelle lichtstrepen loodrecht op de richting van de naad. Een CCD-camera neemt het streepjespatroon op. Het videosignaal afkomstig van de camera wordt voorbewerkt m.b.v. een analoog pulsbreedtefilter in realtime. Het resultaat hiervan is een videosignaal met daarin alleen het streepjespatroon.
Het voorbewerkte videosignaal wordt aangeboden aan de zogenaamde patroonherkenningsmodule - het onderwerp van dit verslag. De taak van deze module is om het streepjespatroon te analyseren met het doel de positie van de naad (overlapnaad) te bepalen. Op de plaatsen in het streepjespatroon waar de overlapnaad zich bevindt, treden vervormingen op. Deze plaatsen worden m.b.v. een DOG-filter (edge-enhancement filter) opgezocht.
De patroonherkenningsmodule is uitgevoerd op een digitale signaalprocessor, n.l. de ADSP-2100 van Analog Devices. De snelheid waarmee patroonherkenning wordt uitgevoerd is 40 msec.

Deckere, W.J.M.A. de, "Design of a pattern recognition module for a fast seam tracking system"
M.Sc. Thesis, Measurement and Control, Electrical Engineering,
Eindhoven University of Technology, jan. 1990.

The group Measurement and Control is working on the project "Fast seam finding and seam tracking". The object is to build a fast seam tracking system. The control system uses sensor information to guide a robot arm along the seams in a piece of work. The seams are being lighted using structured light. The light pattern is sensed with a CCD-camera. The light pattern contains deformations due to geometric form of the seamline.
The image processing system consists of a number of modules. One of these is the pattern recognition module. This module checks whether the deformations correspond with the seam in the object. The checking is done using a DOG-filter (edge enhancement filter). The module generates a list of coordinates that describe the position of the seam.
The patern recognition module is implemented on a ADSP-2100 digital signal processor (Analog Devices). The seam is found within 40 msec.

Voorwoord

Via deze weg zou ik een aantal mensen willen bedanken die hebben meegewerkt aan de voltooiing van mijn afstudeerproject. In het bijzonder noem ik prof. ir. F. Kylstra en ir. R. van Vliet. Ik wil hen bedanken voor de gelegenheid die zij mij hebben geboden om mee te werken aan het onderzoek naar beeldverwerkingstechnieken voor robotsystemen dat op de vakgroep Meten en Regelen wordt verricht.

Mijn afstudeerwerk vormt een deel in het project "snel naadzoeken en naadvolgen". Andere deelprojecten werden gelijktijdig door studenten en medewerkers van de vakgroep Meten en Regelen uitgevoerd. Meewerken aan een groot project als dit, is op zich een zeer leerzame ervaring. Om een dergelijk project te laten slagen is het noodzakelijk dat er onderling goede afspraken worden gemaakt. Zo kreeg men niet alleen te maken met de de technische kant van het project maar ook met organisatorische zaken.

Ik zou graag alle studenten en medewerkers willen bedanken voor de steun die zij mij gegeven hebben bij mijn werkzaamheden. Het was mij een groot genoegen om met hen te hebben mogen samenwerken. Ik wens degenen, die het project zullen voortzetten, veel succes toe.

Inhoudsopgave

1. Inleiding

Bij de vakgroep Meten en Regelen van de faculteit Elektrotechniek en Informatietechniek aan de Technische Universiteit te Eindhoven is in 1984 een project gestart, genaamd "Snel naadzoeken en naadvolgen". Dit project wordt gesubsidieerd door de Stichting Technische Wetenschappen.
In het kader van dat project wordt gewerkt aan de ontwikkeling van een beeldverwerkingssyteem dat een robot in staat moet stellen om naden in een werkstuk met een grote snelheid en nauwkeurigheid te volgen.

Men is bij Volvo geïnteresseerd in de resultaten van dit onderzoek. De bij dit onderzoek ontwikkelde methoden zouden eventueel kunnen worden gebruikt om een naadvolgsysteem te ontwikkelen, dat kan worden ingezet in de lakstraat in de autofabriek van Volvo in Born. Om corrossie aan autocarrosseriën te voorkomen worden de plaatsen, waar de verschillende metalen onderdelen elkaar overlappen (overlapnaad), afgedicht met een beschermingslaagje. Op dit moment worden dit laagje nog met de hand aangebracht maar men hoopt dit proces in de toekomst te automatiseren.

Het project "Snel naadzoeken en naadvolgen" bestaat uit een aantal deelprojecten. Eén van deze deelprojecten is "Ontwerp van een patroonherkenningsmodule voor een snel naadvolgsysteem". Dit deelproject is tevens het afstudeerproject van mijn doctoraalstudie. Mijn werkzaamheden aan dit project zijn beschreven in dit verslag.

De structuur van dit verslag is als volgt.
Na een korte inleiding over het afdichten van naden in carrosseriqn bij Volvo wordt een lijst met systeemspecificaties gegeven voor de te realiseren sensorgestuurde naadvolgrobot.
In het volgende hoofdstuk zijn een aantal beeldverwerkingstechnieken, bekend uit de literatuur, op een rijtje gezet.
Het beeldverwerkingssysteem van het naadvolgsysteem is opgebouwd uit een aantal modules. Deze worden elk afzonderlijk besproken in hoofdstuk 4.
De module, waar in dit verslag speciale aandacht aan zal worden besteed, is de zogenaamde patroonherkenningsmodule. Deze module is zelf ook weer opgebouwd uit een aantal deelmodules. In hoofdstuk 5 worden deze deelmodules beschreven.
Tenslotte volgen de conclusies en een aantal aanbevelingen.

2. Probleemstelling

2.1. Het afdichtproces

Volvo Car B.V. (Nederland) produceerde in 1989 in totaal 145 duizend personenauto's. Veertig duizend auto's hiervan behoren tot de zogenaamde 300 serie, negentig duizend tot de 440 serie en vijftien duizend tot de 480 serie.
Alle auto's werden geproduceerd in de autofabriek in Born (Limburg). Het fabricageproces is opgedeeld in vier productiestappen.
In de eerste stap worden de verschillende metalen onderdelen van de auto geperst uit plaatstaal. In de volgende stap wordt uit deze onderdelen de carrosserie opgebouwd. De metalen onderdelen worden samen gebracht en aan elkaar gepuntlast. In de derde stap wordt de 'kale' carrosserie voorzien van beschermingslagen om corrosie van het plaatstaal te voorkomen. In de laatste stap worden de carrosserie en het onderstel met daarop de motor samengebracht en volgt de montage van verlichting, bekleding, instrumentatie, etcetera.
Elk van de vier productiestappen wordt uitgevoerd op één van de volgende afdelingen:

- Persstraat
- Afdeling carrosseriebouw
- Lakstraat
- Afdeling assemblage

Figuur 2.1 toont de productielijn met de vier verschillende afdelingen.

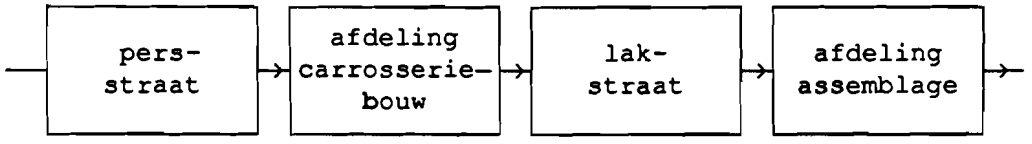


fig. 2.1: Produktielijn Volvo Car Nederland.

D.m.v. automatisering probeert men te besparen op de produktiekosten en probeert men tevens de kwaliteit van het produkt te verbeteren. De persstraat en de afdeling carrosseriebouw zijn reeds ver geautomatiseerd. Op dit moment worden de mogelijkheden onderzocht om een aantal

bewerkingen in de lakstraat te automatiseren.

Hieronder zijn de beschermingslagen genoemd die achtereenvolgens in de lakstraat op de carrosseriën worden aangebracht:

- -Electro coating
- -PVC afdeklaagje voor de overlapnaden aan de onderkant van de carrosserie en rubberachtige substantie voor U-vormige naden aan de voor- en achterzijde van de carroserie
- -Under body coating
- -Filler
- -Grondlaag
- -Laklagen

Nadat olie en vet van het metaal zijn verwijderd wordt de carrosserie ondergedompeld in een verfbad voor het aanbrengen van de eerste laag, n.l. de electro coating. Tussen het bad en de carrosserie staat een spanningsverschil van ongeveer 500 Volt. Onder invloed van het elektrisch veld bewegen de verfdeeltjes (dipolen) zich naar het metaal van de carrosserie.
Op plaatsen waar twee stukken plaatstaal elkaar overlappen ontstaat een zogenaamde overlapnaad. Beide stukken plaatstaal worden niet over de gehele lengte gepuntlast. Hierdoor kunnen tussen twee stukken plaatstaal ruimten ontstaan waarin vocht en vuil gemakkelijk kunnen doordringen. Het plaatstaal zal op die plaatsen snel corroderen. Het probleem van roestvorming is het grootst bij de naden aan de onderzijde van de carrosserie. Om deze plaatsen extra te beschermen dienen daar de naden te worden gevuld met een rubberachtige substantie of dienen deze te worden afgedekt met een laagje PVC. De rubberachtige substantie wordt m.b.v. een kitspuit in de naden gelegd. De PVC wordt aangebracht volgens het airless sealing procedé. Hierbij wordt de PVC m.b.v. een plat mondstuk onder hoge druk over de naden gespoten (figuur 2.2). Het vullen en afdekken van de naden gebeurt op dit moment nog met de hand.
De carrosserie gaat vervolgens naar de cabine waarin twee robots staan opgesteld die de gehele onderkant van de carrosserie voorzien van een zogenaamde under body coating (UBC). Deze UBC is een dikke laag PVC die dient om schade door steenslag te voorkomen.
Tenslotte worden de de naden aan de binnenzijde van de carrosserie afgedekt met een PVC laagje - filler genaamd - en worden de grondlaag en een aantal laklagen aangebracht.

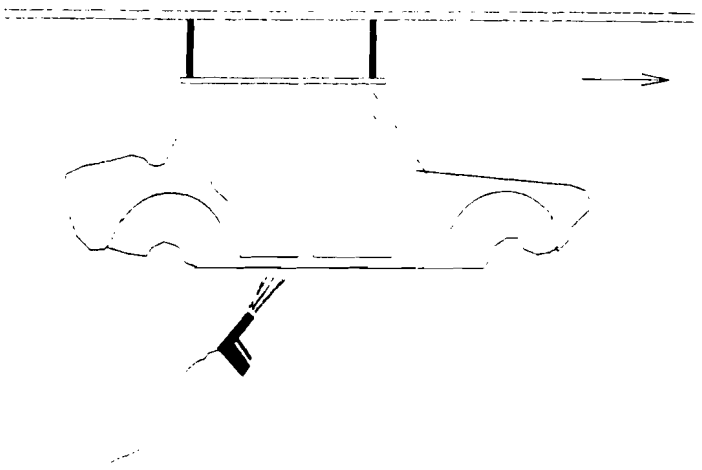


fig 2.2: Airless sealing.

We concentreren ons op de bewerkingsstap waarbij de overlapnaden aan de onderzijde van de carrosserie met een laagje PVC worden afgedekt. Figuur 2.3 toont een dwarsdoorsnede van een overlapnaad. Het plaatstaal heeft een dikte van 0.8 mm. Over de naad wordt een laagje PVC gespoten met een breedte van 4 cm. De maximale onnauwkeurigheid waarmee de werknemers in de lakstraat de PVC-laagjes op de carrosseriën aanbrengen blijkt in de praktijk gelijk te zijn aan 1 cm. De breedte van de spuitmond is zodanig groot gekozen dat bij deze maximale positioneringsonnauwkeurigheid de naad toch geheel wordt afgedekt.

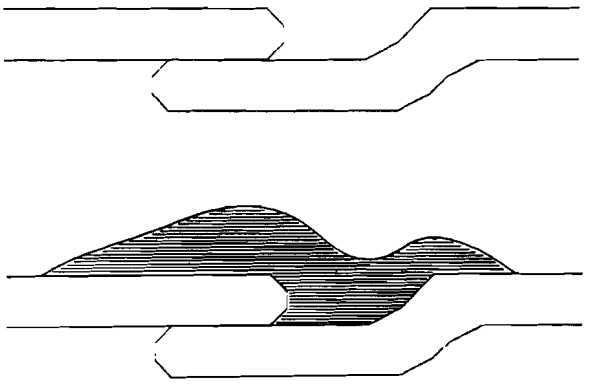


fig. 2.3: Geometrie van de overlapnaad.

Indien de spuitmond met een grotere nauwkeurigheid boven de naad kan worden gepositioneerd kan de breedte van het PVC laagje kleiner worden gekozen om materiaal te besparen. Per auto wordt bij deze bewerking een hoeveelheid PVC gebruikt waarvan de kosten ƒ 8,- bedragen. Bij een productie van 100.000 auto's per jaar levert een reductie van de hoeveelheid PVC met de helft een besparing op van ƒ 400.000,- per jaar.

Men vraagt zich bij Volvo of de automatisering van het afdichtproces, dat in deze paragraaf werd beschreven, met de huidige stand van de techniek mogelijk is. Volvo is daarom geïnteresseerd in het project *"snel naadzoeken en naadvolgen"* dat in 1984 bij de vakgroep Meten- en Regelen van de faculteit Elektrotechniek aan de Technische Universiteit in Eindhoven. Dit project werd gestart met steun van de Stichting Technische Wetenschappen (STW).

Een speciale gebruikersgroep werd opgericht voor bedrijven die belang stelling hebben voor het onderzoek dat in het kader van het project *"snel naadzoeken en naadvolgen"* bij de vakgroep Meten en Regelen wordt verricht.
Naast Volvo hebben drie andere bedrijven in deze groep plaats genomen. Dit zijn: Philips (afdeling CFT, Centrum voor Fabricage Technologie), Pandata (automatiseringsbureau) en Kuka (producent robotsystemen).

2.2. Het naadvolg-robotsysteem: ontwerpeisen

Eén van de doelstellingen van het project is de technische realisatie van een prototype van een automatisch naadvolg-robotsysteem. Het afdichtproces bij Volvo is een mogelijke toepassing. Dit naadvolgsysteem dient in staat te zijn om in een werkstuk de naden te volgen met een nauwkeurigheid van 1 mm. De totale lengte van de naden op de carrosseriën bij Volvo bedraagt 22 m. Op dit moment worden deze naden afgedekt met een laagje PVC door vier personen binnen 1,5 minuut.
Indien deze taak zou worden overgenomen door een automatisch naadvolgsysteem dan dient dit systeem de spuitmond met een gemiddelde snelheid van ongeveer 0,25 $^{m}/_{sec}$ langs de naden te bewegen. Gekozen werd voor de maximum snelheid van 0,5 $^{m}/_{s}$.

Bij de ontwerwerpafdeling van Volvo heeft men de ligging van de naden op de carrosserie m.b.v. een verzameling coördinaten vastgelegd. Gegeven deze verzameling coördinaten kan een baan berekend worden waarlangs de spuitmond moet worden bewogen.
In de praktijk zal deze baan de naden in de carrosserie niet exact overlappen. Op een aantal plaatsen zal de afwijking tussen de baan en de naden zelfs groter zijn dan de vereiste positioneringsnauwkeurigheid van 1 mm.
Dit heeft twee oorzaken. De eerste oorzaak is dat de carrosserie niet nauwkeurig is gepositioneerd t.o.v. het transportsysteem. Dit kan worden verbeterd door gebruik te maken van een mechanische positione-

ringsinrichting, echter dit soort oplossingen is meestal duur. Een andere oplossing is om gebruik te maken van een *sensor* om de positie van de carrosserie t.o.v. het transportsysteem te bepalen. Gegeven deze positie voert het besturingssysteem een correctie uit op de voorgeprogrammeerde baan.
De tweede oorzaak is dat, wanneer de metalen delen worden gepuntlast, er in het algemeen vervormingen optreden in het werkstuk. Om de spuitmond toch met een nauwkeurigheid van 1 mm boven het werkstuk te manoevreren is informatie nodig over de exacte positie van de naad. Door op het naadvolgsysteem een *sensor* te monteren, die met de spuitmond meebeweegt, kan de exacte positie van de naad worden opgenomen. In het geval dat de gemeten positie afwijkt van voorgeprogrammeerde baan dient de spuitmond te worden bijgestuurd.

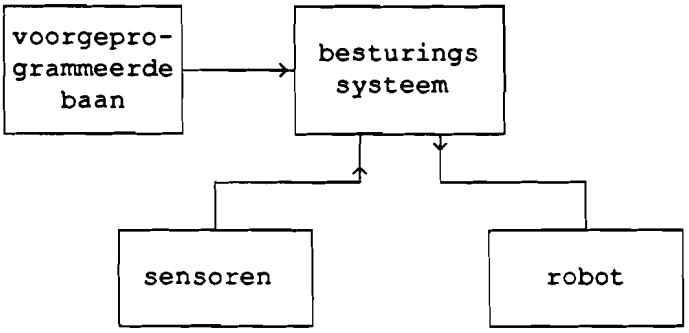


fig. 2.4: Naadvolgsysteem, uitgerust met sensoren.

Samen met de gebruikersgroep van het project werd besloten om een prototype van een snel *sensorinteractief* naadvolgsysteem te bouwen. De nadruk zou hierbij vooral komen te liggen op het ontwerp van de sensoren en hoe deze sensoren kunnen worden geïntegreerd in het besturingssysteem van de robot.
Als prototype koos men voor een cartesische robot - ook wel portaalrobot genoemd - die is uitgerust met lineaire motoren langs de x-, y- en z-as (figuur 2.5). De reden waarom voor dit type robot werd gekozen is dat hiervoor een eenvoudiger (en dus sneller) besturingssysteem ontworpen kan worden dan voor een robot met draaiende gewrichten. De besturing van drie cartesische assen is immers eenvoudiger dan de besturing van een aantal t.o.v. elkaar roterende gewrichten.

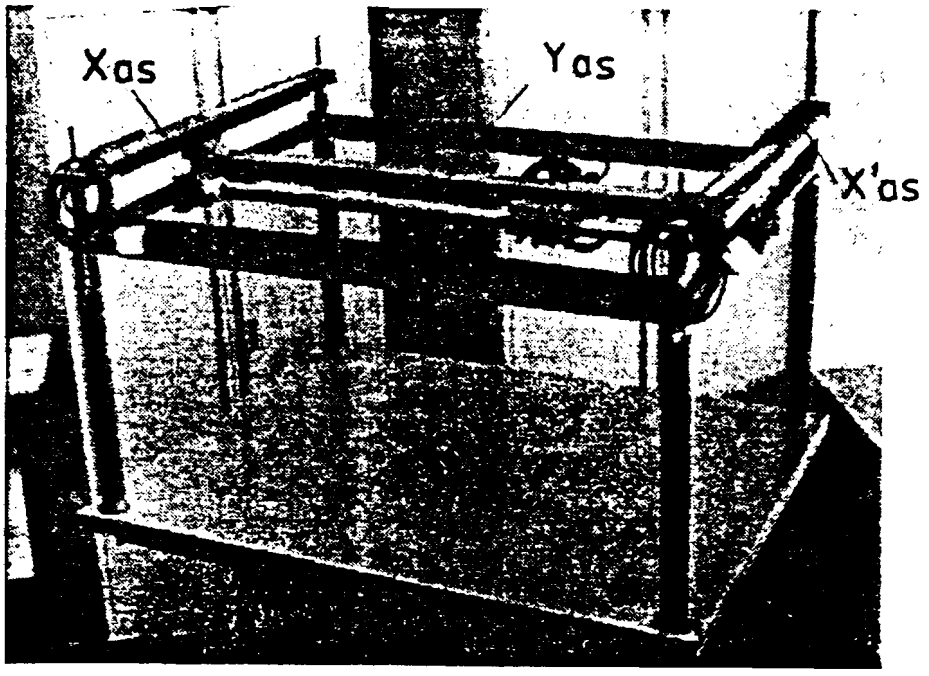


fig. 2.5: De cartesische robot

De sensor op de robotarm zal worden voortbewogen met een maximum snelheid van 0.5 $^{m}/_{s}$. Verder werd gesteld dat de sensor de positie van de naad met nauwkeurigheid van 1 mm moet kunnen bepalen in een gebied van 5x5x5 cm^3. De cyclustijd van het robotsysteem moet dus kleiner zijn dan of gelijk aan 100 msec. Binnen deze tijd dient de beeldinformatie te worden vertaald naar bewegingen van de robot. Het aantal berekeningen, dat nodig is voor het verwerken van de sensordata, moet zo klein mogelijk worden gehouden.

Onderaan de carrosserie wordt plaatstaal gebruikt met een dikte van 0.8 mm. De naadsoort die daar het meest voorkomt is de overlapnaad.

In tabel 2.1 zijn de ontwerpeisen voor een snel naadvolgsysteem samengevat.

Tabel 2.1: Ontwerpeisen naadvolgsysteem.

Dikte plaatstaal	0,8 mm
Naadvorm	overlapnaad
Hoogteverschillen bij de naad	0,8 mm (min)
Positioneernauwkeurigheid t.o.v. de naad	1 mm
Totale lengte van de naden	±22 m
Maximale bewerkingstijd	1,5 minuut
Gemiddelde snelheid spuitmond	0,25 m/s
Maximale snelheid spuitmond	0,5 m/s

3. 3-Dimensionale vormanalyse

3.1. Inleiding

Er dient een systeem te worden ontworpen waarmee in korte tijd de positie kan worden bepaald van naden in objecten zoals een carrosserie. De naden in de carrosserie treden op op plaatsen waar twee stukken metaal elkaar overlappen. De geometrie van zo'n naad is weergegeven in figuur 3.1.

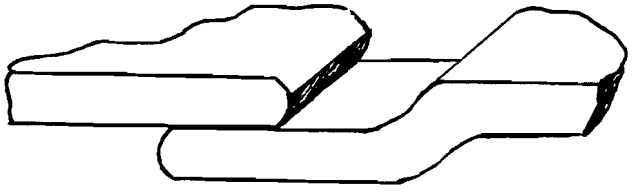


fig. 3.1: Geometrie van een naad in de carrosserie.

Er zijn een aantal technieken bekend waarmee de 3-dimensionale vorm van een object kan worden bepaald. Strand [13] noemt er een aantal waarbij gebruik wordt gemaakt van niet-tactiele sensoren. Hij deelt deze technieken in in de volgende vier categoriën:

- 'time-of-flight' technieken;
- interferometrische technieken;
- diffractie technieken.
- geometrische technieken;

Tot de eerste categorie behoren de zogenaamde *time-of-flight* technieken. Bij deze technieken worden naar het object korte geluidspulsen of elektromagnetische pulsen uitgezonden, waarna de reflectie van het object wordt opgevangen. De tijd tussen het zenden van de puls en het ontvangen van de reflectie is bepalend voor de afstand tot het oppervlak van het object. Voorbeelden, waarin deze technieken worden toegepast, zijn radar en ultrasone echografie.

Tot de volgende twee categoriën behoren zogenaamde *interferometrische* en *diffractie* technieken. Om de 3-dimensionale vorm van het object te bepalen wordt hierbij gebuik gemaakt van speciale eigenschappen van licht, zoals interferentie en breking. Voor voorbeelden wordt verwezen naar het artikel van Strand [13].

Strand noemt als laatste de categorie van de *geometrische* technieken. Deze hebben een aantal belangrijke voordelen t.o.v. de andere genoemde technieken. Zo ligt meestal het concept van de techniek vrij voor de hand en daarmee hangt samen dat ze eenvoudig kunnen worden geïmplementeerd. We zullen ons daarom speciaal op deze categorie concentreren.

Bij geometrische technieken wordt onderscheid gemaakt tussen technieken met *actieve* en technieken met *passieve* belichting.
Een voorbeeld van een techniek met passieve belichting is *stereoscopie*. De 3-dimensionale vorm van een object kan namelijk worden gereconstrueerd uit één paar opnames die vanuit twee verschillende ooghoeken zijn gemaakt. Hierbij hoeft het het object niet op een speciale manier te worden belicht, vandaar de naam passieve belichting. Het omgevingslicht zorgt hier voor de belichting van het object.
Passieve belichting heeft als nadeel dat er ingewikkelde rekentechnieken nodig zijn om de 3-dimensonale vorm van het object uit de beeldinformatie te reconstrueren. Dit is niet het geval bij geometrische technieken met actieve belichting. Het object wordt belicht m.b.v. *gestructureerd licht* waarbij min of meer direct de 3-dimensionale vorm van het object te voorschijn komt. Bij gestructureerd licht moet men denken aan technieken waarbij het object wordt belicht m.b.v.:

- een lichtstraal
- een lichtvlak
- meerdere lichtvlakken

Voorbeelden worden getoond in figuur 3.2. Op de plaats waar de lichtstraal het oppervlak van het object snijdt ontstaat een lichtpunt. Op de plaatsen waar een lichtvlak het oppervlak snijdt ontstaat een lichtstreep.

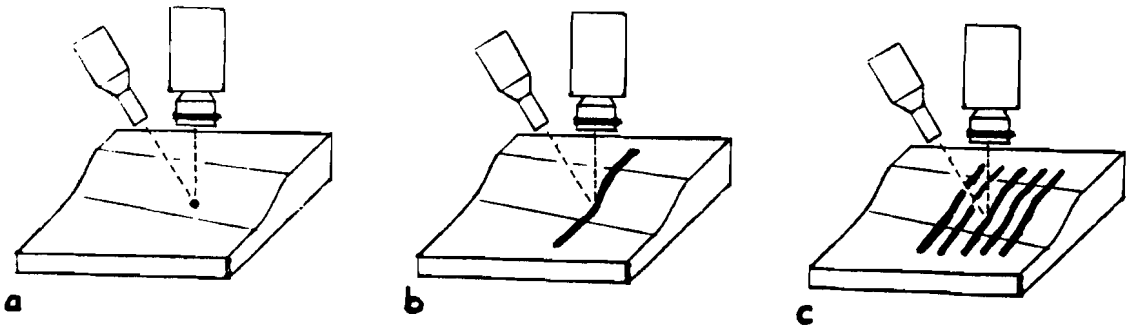


fig. 3.2:geometrische technieken met actieve belichting:
a) belichting m.b.v. een lichtstraal
b) belichting m.b.v. één lichtvlak
c) belichting m.b.v. meer lichtvlakken

De hoek waaronder het lichtpatroon op het object wordt geprojecteerd verschilt van de hoek waaronder naar het object wordt gekeken. M.b.v. driehoeksmetingen (triangulatie) kan de 3-dimensionale vorm van het object worden gereconstrueerd. De opname van het lichtpatroon kan worden gemaakt m.b.v. een standaard camera, zoals de CCD-camera beschreven in appendix A.

Conclusie:

In deze paragraaf werden verschillende technieken besproken die kunnen worden gebruikt om de 3-dimensionale vorm van een werkstuk te analyseren. Figuur 3.3 toont een overzicht.
Het doel van het project is om een beeldverwerkingssysteem te ontwikkelen dat in staat is om *realtime* de positie van een naad in werkstuk te bepalen. Een geometrische techniek met actieve belichting is het meest geschikt. Immers een dergelijke techniek levert direct informatie over de 3-dimensionale vorm van het werkstuk zonder dat er veel rekenwerk moet worden uitgevoerd.

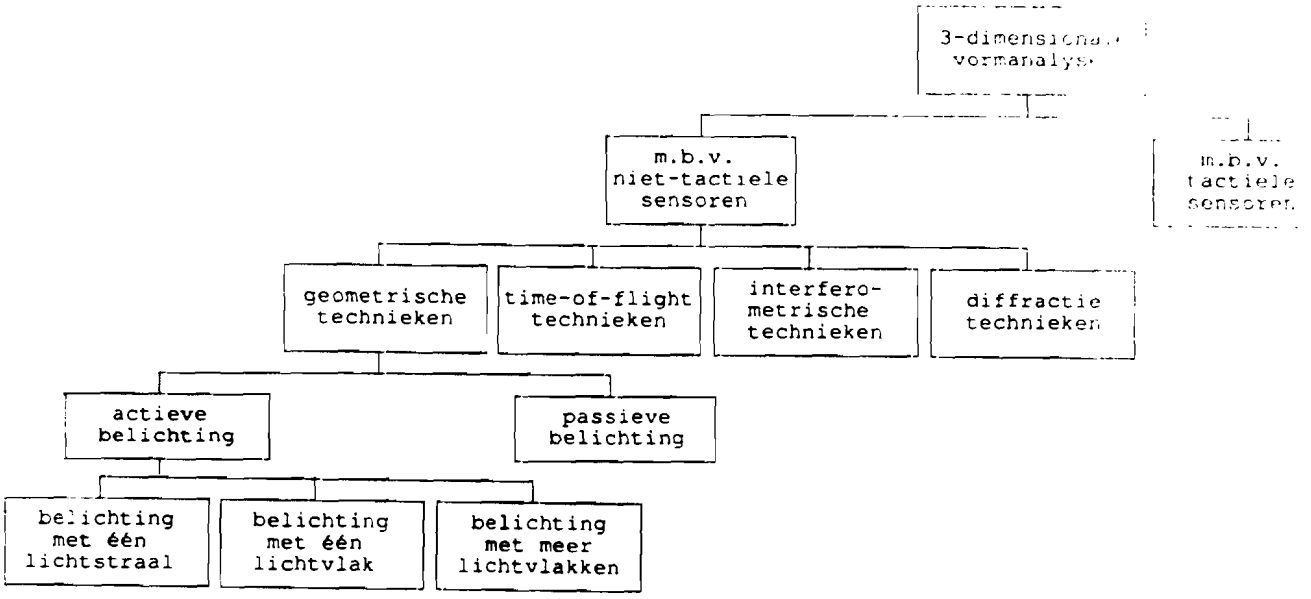


fig. 3.3: Schema 3-dimensionale vormanalyse.

3.2 Beeldverwerkingstechnieken

Uit de literatuur zijn een aantal voorbeelden bekend van realtime naadvolgsystemen waarbij geometrische technieken worden gebruikt om de 3-dimensionale vorm van het werkstuk te analyseren. Twee voorbeelden hiervan worden besproken in de paragrafen 3.2.1 en 3.2.2. Het eerste voorbeeld werd eerder beschreven door W.F. Clocksin [6 en 7]. Het tweede voorbeeld door R. Niepold [10].

Bij systeem van Clocksin wordt een lichtpatroon op het werkstuk geprojecteerd dat bestaat uit één lichtstreep. Bij het systeem van Niepold worden drie lichtstrepen geprojecteerd.
Het lichtpatroon op het werkstuk wordt opgenomen m.b.v. een standaard videocamera. M.b.v. een filterbewerking kunnen de plaatsen, waar het lichtpatroon in het videobeeld optreedt, geaccentueerd worden terwijl storingen in het beeld worden onderdrukt.
Stel dat deze filterbewerking wordt uitgevoerd door een computer. Het videosignaal wordt eerst gedigitaliseerd - bijvoorbeeld m.b.v. een framegrabber - en daarna wordt de digitale beeldinformatie overgebracht naar het geheugen van de computer. De hoeveelheid beeldinformatie, die de computer te verwerken krijgt, is ongeveer gelijk aan

260.000 byte (gesteld dat het videobeeld is opgebouwd uit 512x512 pixels en dat grijstinten kunnen worden gerepresenteerd m.b.v. één byte). Gezien deze grote hoeveelheid beeldinformatie duurt het relatief lang voordat de filterbewerking door de computer is uitgevoerd.

Het bijzondere bij de beeldverwerkingssystemen, beschreven door Clocksin en Niepold, is dat de filteroperatie wordt uitgevoerd m.b.v. analoge filtertechnieken. Het uitvoeren van een filterbewerking op het videosignaal m.b.v. een analoog filter gaat *realtime*. Dit is altijd *sneller* dan wat met een microprocessor kan worden bereikt.

3.2.1. De methode van Clocksin

Clocksin [6 en 7] beschrijft een naadvolgsysteem voor het automatisch booglassen van dunne metalen platen. Op een robotarm zijn een lastoorts en een camera/projector combinatie gemonteerd. De robotarm wordt voortbewogen langs een baan die in de besturingscomputer van de robot is voorgeprogrammeerd. De camera/projector combinatie wordt getoond in figuur 3.4a.

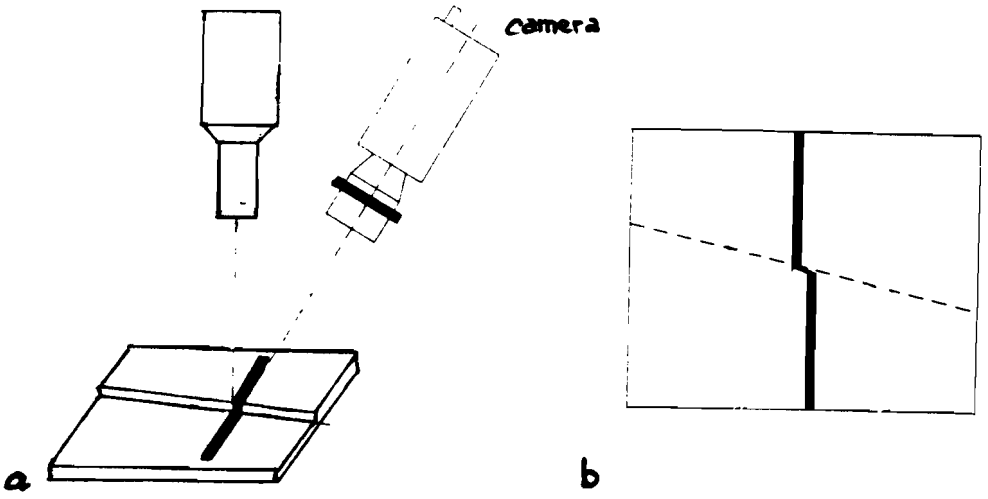


fig. 3.4:a) Camera/projector combinatie;
b) Snijlijn lichtvlak/materiaaloppervlak.

Boven het werkstuk hangt een CCD camera waarvan de camera-as loodrecht staat op het oppervlak van het werkstuk. Een dunne laserbundel zwaait zeer snel heen en weer over het materiaaloppervlak. Het vlak dat door de laserstraal wordt beschreven maakt een kleine hoek met de camera-as. Op de plaatsen waar het lichtvlak het materiaaloppervlak snijdt ontstaat een lichtstreep. Deze lichtlijn wordt afgebeeld op het lichtgevoelige gedeelte van de CCD-camera. Camera en projector zijn zodanig

opgesteld dat de lichtstreep ongeveer loodrecht staat op de beeldlijnen van de camera (figuur 3.4b).

fig. 3.5: Beeldlijn.

Figuur 3.5 toont het videosignaal van één beeldlijn waarin we een aantal signaalpulsen met verschillende pulsduur kunnen onderscheiden. Er is één signaalpuls bij waarvan de pulsduur correspondeert met de breedte van de lichtstreep in het camerabeeld. De andere pulsen zijn op te vatten als storingen die uit het videosignaal weggefilterd dienen te worden.

Clocksin ontwierp een pulsbreedtefilter dat pulsen uit de beeldlijnen verwijdert die een pulsduur hebben die niet overeen komt met de breedte van de lichtstrepen. Het pulsbreedtefilter bestaat uit een DOG-filter (looptijdfilter) en een drempeldetector. De impulsresponsie van dit filter is weergegeven in figuur 3.6. De responsie is maximaal wanneer op de ingang een puls verschijnt die even breed is als het positieve deel van de impulsresponsie.

fig 3.6: Impulsreponsie DOG filter.

Van de gedetecteerde pulsen worden de beeldcoördinaten bepaald en deze worden doorgegeven aan het patroonherkenningssysteem. Dit systeem analyseert de vorm van de lichtstreep om de positie, vorm en eventueel andere kenmerken van de naad te bepalen. De resultaten worden aangebo-

den aan het besturingssysteem die deze gebruikt om een robotarm te sturen. De verschillende bewerkingsstappen worden schematisch weergegeven in figuur 3.7.

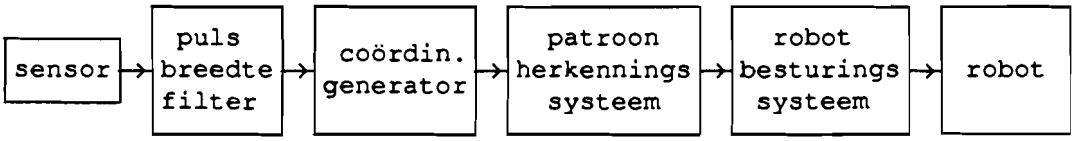


fig. 3.7: Blokschema naadvolgsysteem volgens Clocksin.

De vormanalyse geschiedt d.m.v. *syntactische patroonherkenning*. Bij deze methode dient de set coördinaten, die aan het patroonherkenningssysteem wordt aangeboden, eerst te worden benaderd d.m.v. een set rechte lijnstukken, de zogenaamde primitieven. Daarna wordt gezocht naar patronen van primitieven die behoren tot de klasse van de gezochte naadvorm.
Voor een meer theoretische beschouwing wordt verwezen naar het artikel van Fu [8].

3.2.2. De methode van Niepold

Niepold [10] beschrijft een naadzoeksysteem waarbij ook gebruik wordt gemaakt van gestructureerd licht om de positie van de naad in het werkstuk te bepalen. Bij dit ontwerp worden drie lichtvlakken onder een hoek van 45° op het werkstuk geprojecteerd. Als lichtbron wordt nu geen laser gebruikt maar een halogeenlamp. De snijlijnen van de lichtvlakken en het materiaaloppervlak staan ongeveer loodrecht op de naad (figuur 3.8).

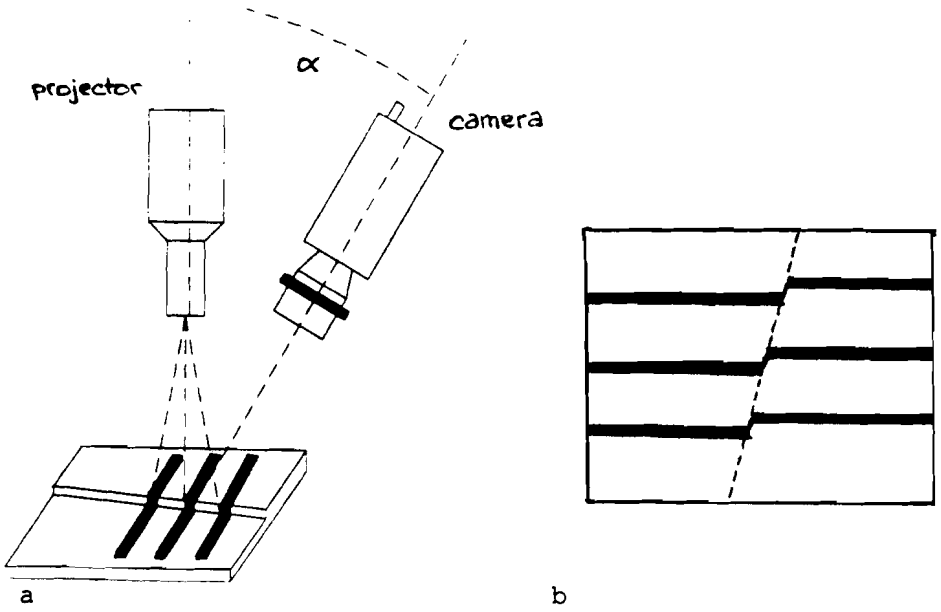


fig. 3.8:a) Camera/projector combinatie;
b) Snijlijn lichtvlak/materiaaloppervlak.

Boven het werkstuk hangt een camera met de camera-as loodrecht op het materiaaloppervlak. De camera is zodanig gedraaid dat de lichtstrepen ongeveer evenwijdig liggen aan de beeldlijnen van de camera. M.b.v. een *edge-detection* filter worden op de beeldlijnen overgangen in de lichtintensiteit bepaald (overgangen van donker naar licht). Deze overgangen treden op op de plaatsen waar de naad zit. De beeldcoördinaten van de plaatsen waar zo'n overgang in het camerabeeld optreedt worden bepaald door de coördinatengenerator. In het camerabeeld kunnen meerdere naden voorkomen. De coördinatenselector dient coördinaten van de gezochte naad te onderscheiden van de coördinaten van de andere naden. In theorie levert het edge-detection filter direct de coördinaten van de naad. Er hoeft verder geen rekenintensief patroonherkenningsalgoritme te worden uitgevoerd. M.b.v. van het door Niepold omschreven ontwerp is men in staat om binnen 30 msec de positie van de naad te bepalen.

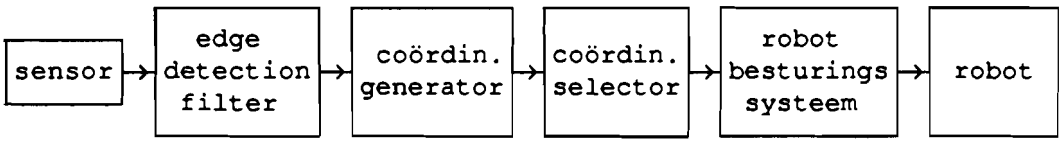


fig. 3.9: Blokschema naadvolgsysteem volgens Niepold.

3.3. Beeldverwerkingsalgoritmen

In de gefilterde beeldinformatie komen patronen voor die karakteristiek zijn voor de gezochte naad. De patroonherkenningsmodule heeft de taak om deze patronen op te sporen. Dit is nodig om de positie van de naad te bepalen.

Clocksin en Niepold beschrijven in hun artikelen globaal het algoritme dat de karakteristieke patronen opzoekt in de gefilterde beeldinformatie. Om beide algoritmen te testen had ik de beschikking over een *general purpose* beeldverwerkingssysteem, n.l. de *Image Processor Series 150/151* van de *firma Imaging Technolgy*. Voor meer informatie over dit systeem verwijs ik u naar appendix B.

Bij de tests op het beeldverwerkingssysteem werd als proefwerkstuk een autoportier gebruikt, dat beschikbaar gesteld door Volvo Car B.V. De naden in dit werkstuk hebben een vergelijkbare geometrie met de naden aan de onderkant van de carrosserie.
Het doel van de tests is om de voor- en nadelen van de methoden, beschreven door Clocksin en Niepold, aan het licht te brengen.

In paragraaf 3.3.1 wordt de algoritme besproken, dat door Clocksin werd ontwikkeld. In paragraaf 3.3.2 komt het algoritme aan de orde, beschreven door Niepold.

3.3.1. De methode van Clocksin

Clocksin maakt gebruik van syntactische patroonherkenning om de positie van de naad in het streepjespatroon te bepalen. Om deze techniek te kunnen gebruiken dient de set coördinatenparen (oftewel streeppunten) eerst te worden benaderd d.m.v. een aantal rechte lijnstukken

(dit is: lineaire approximatie).

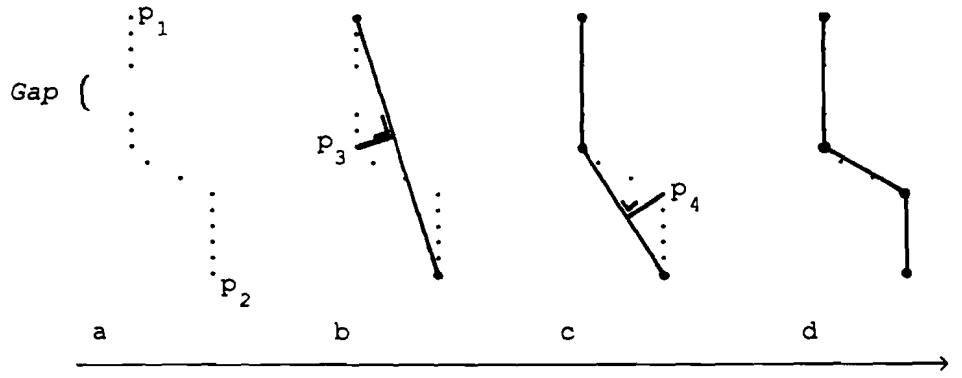


fig. 3.10: Set coördinatenparen benaderd d.m.v rechte lijnstukken (lineaire approximatie).

Figuur 3.10 toont een methode om de verzameling streeppunten, die de lichtstreepjes beschrijven, te benaderen d.m.v. een aantal rechte lijnstukken.
Tussen de punten p1 en p2 wordt een rechte lijn getrokken. Deze lijn geldt als eerste grove benadering. Indien er in de verzameling streeppunten een punt voorkomt waarvan de afstand tot de lijn groter is dan een bepaalde drempel dan is de benadering nog niet goed genoeg. In dat geval wordt het punt opgezocht dat een maximale afstand heeft tot de eerste benadering: n.l. het punt p3. Bij dit punt wordt de eerste benadering opgesplitst in twee nieuwe stukken, n.l. de lijnstukken p1-p3 en p3-p2.
Dit proces zet zich zo voort totdat in de verzameling geen enkel punt meer is te vinden waarvan de afstand tot de set rechte lijnstukken groter is dan de drempel.

In figuur 3.10 ontbreken een aantal streeppunten in de lichtstreep (aangegeven met: *Gap*). Merk op dat dit geen problemen oplevert voor de algoritme. Bij de lineaire approximatie worden deze onderbrekingen opgevuld.

De bovenstaande recursieve algoritme is geïmplementeerd op de Image Processor Series 150/151 van Imaging Technology (appendix B). Bij tests bleek dat het algoritme gevoelig is voor storingen in de lichtstreep. Indien een verzameling streeppunten moet worden benaderd zoals weergegeven in figuur 3.11 dan zijn er een groot aantal stappen nodig voordat deze is benaderd d.m.v. een set rechte lijnstukken. Een dergelijke storing kan ontstaan wanneer op het werkstuk een metaalsnipper ligt.

fig. 3.11: Dataset met storingen.

Dit probleem kan worden opgelost door vooraf een filterbewerking uit te voeren waarbij dit soort storingen uit de lichtstreep worden verwijderd.

Wanneer de lichtstrepen complexere vormen aannemen, bijvoorbeeld omdat er meer naden in het camerabeeld voorkomen, dan zal het lang duren voordat een goede benadering voor de verzameling coördinaten is gevonden.

3.3.2. De methode van Niepold

Volgens Niepold levert de filteroperatie, in het door hem beschreven systeem, direct de coördinaten van de gezochte naad.
Op het beeldverwerkingssysteem werd getest of de methode van Niepold geschikt is om de naad op te zoeken in een werkstuk, zoals het autoportier van Volvo Car B.V.

Bij de tests bleek dat de methode van Niepold slechts in beperkte situaties kan worden toegepast. Wanneer het oppervlak, waarin de naad ligt, niet gekromd is dan voldoet de methode van Niepold uitstekend. Echter wanneer het oppervlak van het werkstuk een complexere vorm heeft, zoals bij het autoportier, dan ontstaan er problemen. Wanneer bijvoorbeeld het oppervlak van het werkstuk enigszins gekromd is dan zullen de lichtstrepen niet meer evenwijdig aan de beeldlijnen liggen. Het gevolg is dat het edge-detection filter punten vindt die niet tot de naad behoren (zie figuur 3.12).

Ik voer hier de term *connectivity* in:

> Twee punten in een camerabeeld hebben *connectivity* wanneer deze verbonden zijn door andere punten. De connectivity van een verzameling beeldpunten die op een lijn liggen is gelijk aan het aantal punten van die lijn. Een geïsoleerd punt heeft een connectivity van 1.

Een eigenschap van de *valse* beeldpunten in figuur 3.&& is dat ze weinig connectivity vertonen. Punten met weinig of geen connectivity dienen dus uit het beeld te worden verwijderd.

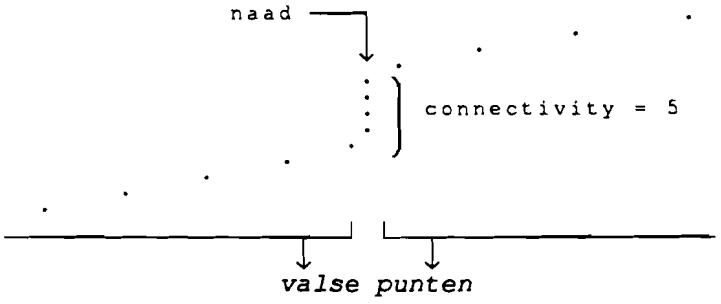


fig 3.12: Detectie van valse punten. De oorzaak hiervan is dat de lichtstreep op het werkstuk niet evenwijdig ligt aan de beeldlijnen van de camera.

Ik heb de Image Processor Series 150/151 een algoritme geïmplementeerd - het zogenaamde *connectivity filter* - waarmee het grootste deel van de *valse* beeldpunten uit het beeld worden verwijderd.
Echter er treedt nog een ander probleem op. Vaak is het plaatstaal in de buurt van een naad gebold. Op dit gebolde oppervlak kunnen reflecties optreden die door het edge-detection filter als een licht-donker overgang worden waargenomen. Afhankelijk van de vorm van het plaatstaal komen parallel aan de naad korte lijnstukken te liggen (figuur 3.13), waarvan de connectivity zodanig groot dat ze niet door het connectivity filter worden verwijderd.

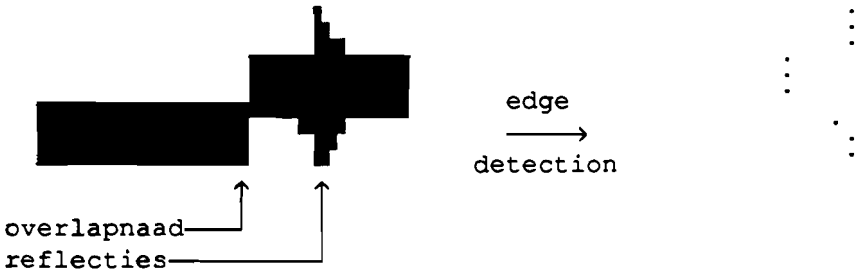


fig. 3.13: Door reflecties op gebolde oppervlakken treden in de buurt van de naad beeldpunten op met een grote connectivity.

Om onderscheid te maken tussen de punten die op de naad liggen en de punten die ontstaan door reflecties kan gekeken worden naar de connectivity van deze beeldpunten. In theorie zal de connectivity van de punten die tot de naad behoren overeenkomen met het sprongetje dat optreedt als gevolg van het hoogteverschil in het metaaloppervlak (figuur 3.14a). Echter dit gaat in de praktijk niet op. Door schaduwwerking bij de naad wordt de connectivity ongeveer gelijk aan de breedte van de lichtstreep (figuur 3.14b).

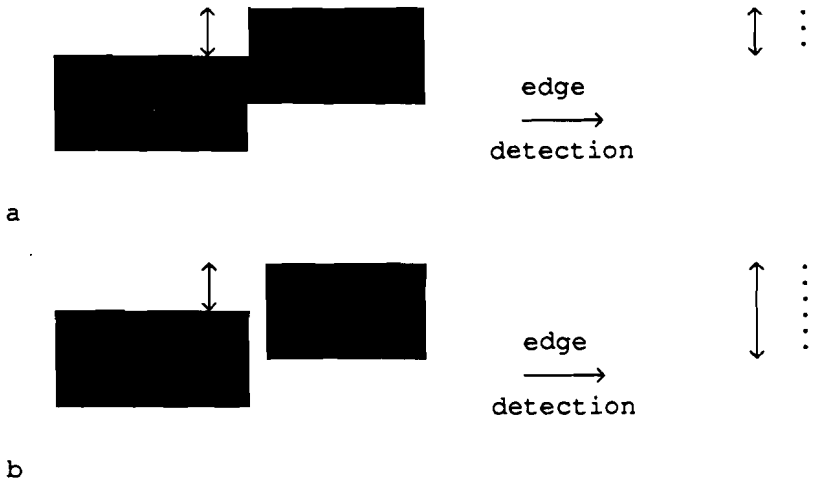


fig. 3.14: a) Connectivity bij een overlapnaad in theorie,
b) Connectivity bij een overlapnaad in praktijk.

Door de hierboven beschreven effecten wordt het ontwerp van een betrouwbaar naadzoeksysteem gebaseerd op de van Niepold bijna onuitvoerbaar. Het grootste probleem is dat, wanneer in de praktijk de edge detection operatie wordt uitgevoerd, de informatie over de vorm van de naad verloren gaat.

Om toch het onderscheid te kunnen maken tussen de punten, die behoren tot de naad, en de punten, die ontstaan door reflecties, zal men gebruik moeten maken van voorkennis over de naad en zijn omgeving. Denk hierbij bijvoorbeeld aan de kans dat er reflecties optreden als gevolg van gebolde oppervlakken in de buurt van de naad.

3.4. Samenvatting en conclusies

In het algemeen wordt bij naadvolgsystemen gebruik gemaakt van gestructureerd licht om de geometrische vorm van het werkstuk te analyseren. Het gestructureerde licht maakt de vorm van het werkstuk meteen zichtbaar.
Om aan de realtime eis te kunnen voldoen wordt de beeldinformatie in het videosignaal m.b.v. analoge filters voorbewerkt.

In de paragrafen 3.2.1 en 3.2.2 werden twee beeldverwerkingstechnieken besproken waarbij gebruik wordt gemaakt van gestructureerd licht (methode van Clocksin en methode van Niepold).
De methode van Niepold is onbetrouwbaar wanneer de naden in een werkstuk met een complexe vorm liggen.
De methode van Clocksin is meer geschikt. Immers hierbij wordt het werkstuk zodanig belicht dat de informatie over de vorm van de naad niet verloren gaat.

Na voorbewerking van het videosignaal houden we een verzameling beeldpunten over die de lichtstreep op het werkstuk beschrijft. In de lichtstrepen zien we vervomingen die karakteristiek zijn voor de gezochte naad. M.b.v. een patroonherkenninsalgoritme wordt de vorm van de lichtstreep geanalyseerd om zo de positie van de naad te bepalen.

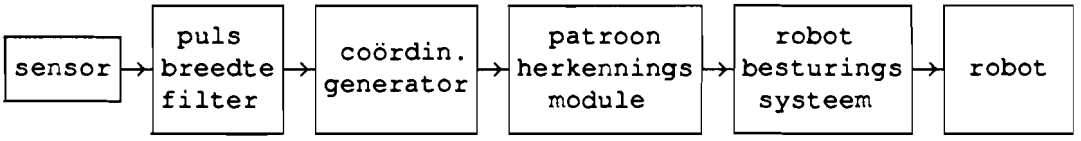


fig. 3.15: Blokschema naadvolgsysteem volgens Clocksin.

Het beeldverwerkingssysteem, dat op de vakgroep Meten en Regelen van de faculteit Elektrotechniek wordt ontwikkeld, heeft ongeveer dezelfde opbouw als het systeem van Clocksin dat wordt getoond in figuur 3.15.
Er zijn echter twee verschillen.
Ten eerste worden bij de belichting van het werkstuk (sensormodule) drie lichtstreepjes gebruikt i.p.v. één lichtstreep.
Ten tweede wordt in het systeem geen gebruik gemaakt van syntactische patroonherkenning. Syntactische patroonherkenning is over het algemeen redelijk betrouwbare techniek, echter zij is rekenintensief. In het nieuwe systeem wordt de patroonherkenning uitgevoerd door de verzameling beeldpunten, die de lichtstrepen beschrijft, te bewerken met een DOG-filter. Het DOG-filter wordt hierbij gebruikt als een zogenaamd edge-detection filter.

In hoofdstuk 4 volgt een beschrijving van de modules waaruit het beeldverwerkingssysteem is opgebouwd.
In hoofdstuk 5 zal nader worden ingegaan op de patroonherkenning m.b.v. het DOG-filter.

4. Het beeldverwerkingssysteem

4.1. Inleiding

Een van de doelstelling van het project *"Snel naadzoeken en naadvolgen"* is om te onderzoeken of een naadvolgsysteem te realiseren is dat voldoet aan de ontwerpeisen gegeven in paragraaf 2.2. Het blokschema van dit systeem is gegeven in figuur 4.1. Het beeldverwerkingssysteem analyseert het beeld van het werkstuk met het doel om de positie van de naad te vinden. Zodra de naad is gevonden worden de coördinaten, die de positie van de naad beschrijven, overgebracht naar het robotbesturingssysteem. Dit systeem zorgt er dan voor de spuitmond met een nauwkeurigheid van 1 mm langs de naad wordt gedirigeerd.

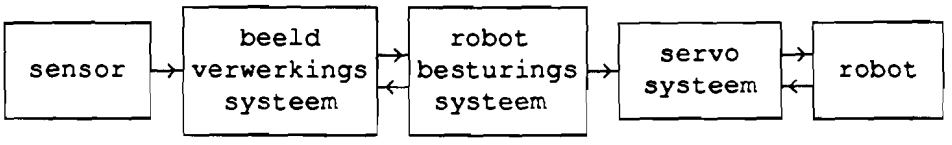


fig. 4.1: Het naadvolgsysteem.

In het vorige hoofdstuk hebben we kennis gemaakt met een aantal beeldverwerkingstechnieken voor het zoeken naar naden in een werkstuk. Bij alle besproken technieken wordt gebruik gemaakt van gestructureerd licht om de geometrische vorm van het werkstuk te analyseren. De hoek, die het streep(jes)patroon maakt t.o.v. de beeldlijnen van de camera, bepaalt de opbouw van het beeldverwerkingssysteem (zie § 3.2.1 tot en met § 3.2.3).

In § 3.4 werd de keuze gemaakt om op het werkstuk een streepjespatroon te projecteren, dat bestaat uit drie lichtstreepjes. Daarnaast wordt de camera zodanig boven het werkstuk gepositioneerd dat de lichtstrepen ongeveer loodrecht op de beeldlijnen in het camerabeeld komen te liggen. In figuur 4.2 is het blokschema gegeven van beeldverwerkingssysteem waarmee de camerabeelden dienen te worden verwerkt.

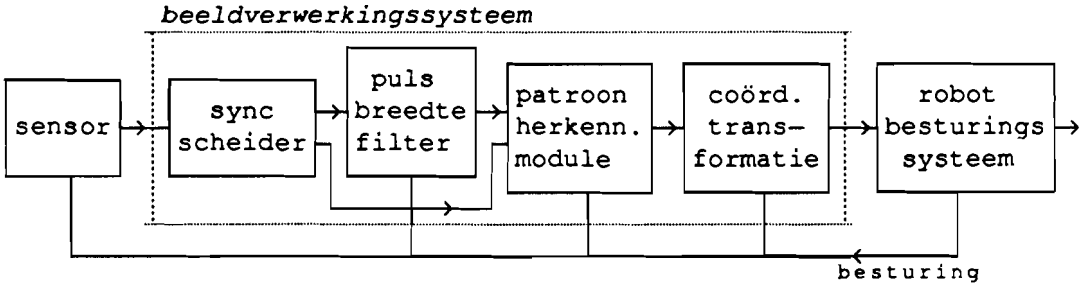


fig. 4.2: Het beeldverwerkingssysteem (.....)

De camera, die bij de experimenten wordt gebruikt, is een camera van de firma HTH. De specificaties van deze camera zijn gegeven in appendix A.
Het beeld dat de camera opneemt is opgesplitst in twee halfbeelden. Deze twee halfbeelden ontstaan door interliniëring. Elk halfbeeld is opgebouwd uit 288 beeldlijnen en één beeldlijn bevat 604 pixels. Het zichtveld van de camera is 6.7 cm hoog en 5 cm breed (de afstand van de camera tot het object is ongeveer 15 cm). Aldus is de resolutie van een halfbeeld in horizontale richting gelijk aan 0.11 mm per pixel en in verticale richting gelijk aan 0.20 mm per pixel.

Om 20 msec tijdswinst te maken wordt alleen het eerste halfbeeld verwerkt. Dit levert geen problemen op voor de nauwkeurigheid waarmee de positie van de naad moet worden bepaald. Immers deze nauwkeurigheid is gelijk aan 1 mm en veel groter dan de resolutie in verticale richting.

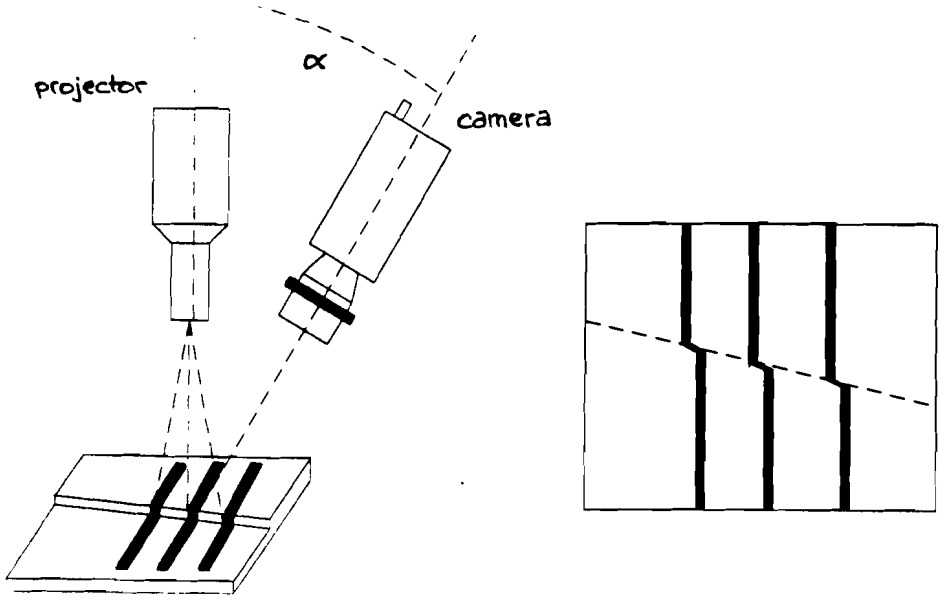


fig. 4.3: Sensor.

Figuur 4.3 toont het sensorsysteem. De hoek α is de hoek die de as van de camera maakt t.o.v. de as van de projector. De verplaatsing Δd in de lichtstreep die de camera waarneemt op de plaats waar de naad zit is afhankelijk van de de dikte D van het metaal en van de hoek α:

$$\Delta d = D \cdot \tan(\alpha)$$

De hoek α is ongeveer 30° groot en de dikte van het plaatstaal bedraagt minimaal 0.8 mm. De verplaatsing Δd is dan minimaal 0.46 mm. Dit komt overeen met 4 pixels (horizontale resolutie).

Op dit moment wordt een projector gebruikt waarvan de scherptediepte kleiner is dan die van de camera. Om deze reden is de projector loodrecht en de camera schuin op het werkstuk gericht.

Het beeldverwerkingssysteem, weergegeven in figuur 4.2, is bestaat uit de volgende modules:

- de sync-scheider
- het pulsbreedtefilter
- de patroonherkenningsmodule
- de coördinatentransformatiemodule

Deze modules zullen achtereenvolgens in de paragrafen 4.1 tot en met 4.5 worden besproken.

4.2. De sync-scheider

De CCD-camera genereert een video-signaal dat voldoet aan de CCIR-norm (paragraaf 3.2). Om te voorkomen dat de synchronisatiepulsen in het videosignaal invloed hebben op de responsie van het pulsbreedtefilter, dienen deze uit het videosignaal te worden verwijderd. Deze taak wordt uitgevoerd door de zogenaamde sync-scheider (figuur 4.4).

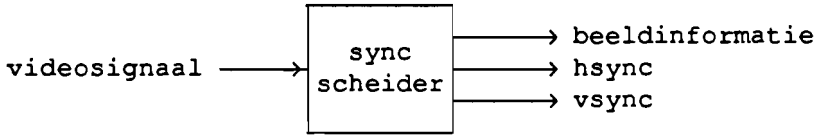


fig. 4.4: De sync-scheider.

De sync-scheider heeft drie uitgangen. Op de eerste uitgang verschijnt het videosignaal zonder de synchronisatiepulsen. De synchronisatiepulsen worden doorgevoerd naar de uitgangen hsync en vsync. Uitgang hsync is actief ten tijde van de horizontale synchronisatie in het videosignaal. Uitgang vsync is actief ten tijde van de verticale synchronisatie.
De signaalniveau's van de genoemde uitgangen en de pulsduur van de hsync en vsync pulsen zijn gespecificeerd in tabel 4.1.

Tabel 4.1: Specificatie uitgangssignalen sync-scheider.

uitgang	signaalniveau	pulsduur
beeldinformatie	analoog, 0 tot 1 Volt	n.v.t.
hsync	digitaal, TTL hoogactief	± 4.7 µsec
vsync	digitaal, TTL hoogactief	± 180 µsec

4.3. Het pulsbreedtefilter

Het werkstuk onder de camera wordt belicht m.b.v. gestructureerd licht. De camera is zodanig opgesteld dat het streepjespatroon ongeveer loodrecht staat op de beeldlijnen. Figuur 4.5 toont de beeldinformatie die op de uitgang van de sync-scheider verschijnt. De blokvormige pulsen in de beeldinformatie corresponderen met de lichtstrepen in het camerabeeld.

fig 4.5 :Beeldinformatie van één beeldlijn
(videosignaal zonder synchronisatiepulsen).

Om de hoeveelheid beeldinformatie te reduceren wordt de beeldinformatie voorbewerkt m.b.v. het pulsbreedtefilter. Dit filter bestaat uit een analoog DOG-filter en een detectieschakeling (figuur 4.6a).
De impulsresponsie van het DOG-filter verloopt ongeveer als het verschil van twee Gaussische krommen waarbij de standaarddeviaties (σ_1 en σ_2) van deze krommen zich verhouden als 1 staat tot 1,6 (figuur 4.6b).
De afkorting DOG staat voor *difference of Gaussians*.

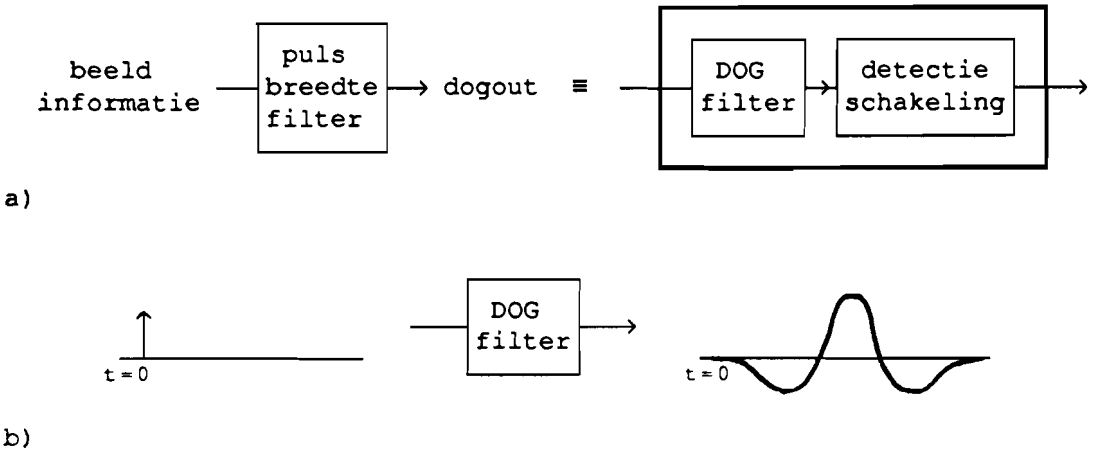


fig. 4.6: a) Het pulsbreedte filter.
b) Impulsresponsie DOG-filter.

De impulsresponsie van het DOG-filter is maximaal wanneer op de ingang een puls verschijnt die ongeveer even breed is als het positieve deel van de impulsresponsie (figuur 4.7).

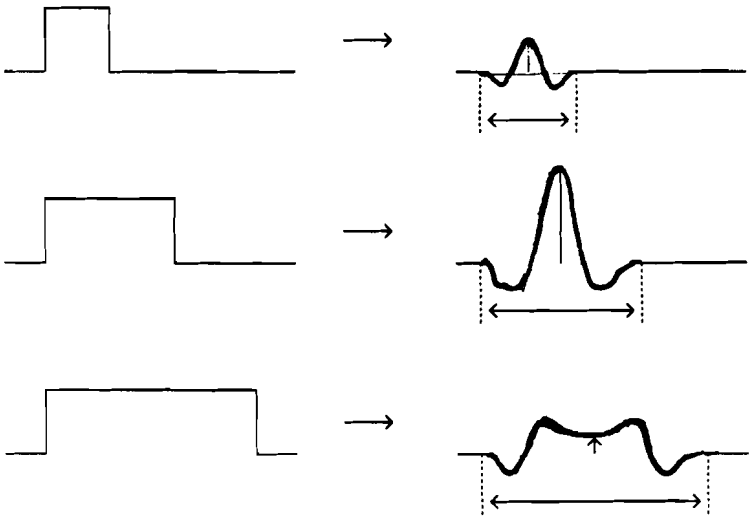


fig 4.7: Responsie op pulsen met verschillende breedte.

Indien de breedte en de grootte van de responsie binnen bepaalde marges liggen wordt door de detectieschakeling een pulsje gegenereerd. Ruis, te smalle en te brede pulsen worden door het filter uit de beeldinformatie verwijderd.
Het pulsje dat door de detectieschakeling wordt gegenereerd voldoet aan de specificatie die is gegeven in tabel 4.2.

Tabel 4.2: Specificatie uitgangssignaal pulsbreedtefilter.

uitgang:	dogout
signaalniveau:	digitaal, TTL hoogactief
indien geen puls gedetecteerd:	0 Volt
indien puls gedetecteerd:	5 Volt, gedurende 40 nsec

4.4. De patroonherkenningsmodule

De gefilterde beeldinformatie wordt aangeboden aan de patroonherkenningsmodule. De taak van deze module is om uit de beeldinformatie de plaats van de naad te bepalen met voorkennis over de vorm van de naad.

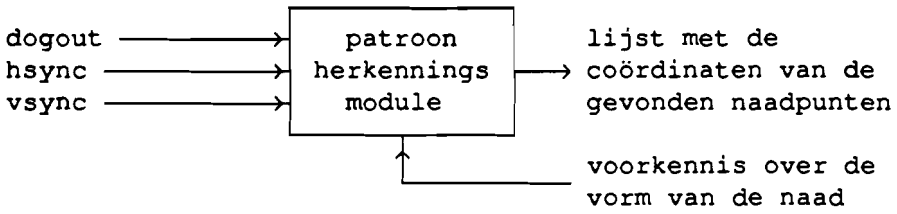


fig. 4.8: Patroonherkenningsmodule.

In deze paragraaf worden een aantal zaken genoemd waarmee rekening moet worden gehouden bij het ontwerp van de patroonherkenningsmodule.

In principe zal de patroonherkenningsmodule verschillende typen naden moeten kunnen herkennen. Echter voorlopig dient de module alleen overlapnaden te kunnen herkennen.
Als gevolg van het hoogteverschil in het materiaaloppervlak, 'ziet' de camera een sprongetje in de lichtstreep. Deze sprong is minimaal 0,46 mm en dat komt overeen met 4 pixels in het camerabeeld.

De vereiste cyclustijd van de robot is 100 msec. D.w.z. dat binnen deze tijd het beeld, dat de camera opneemt, moet worden vertaald naar bewegingen van de robot. De eis die aan het beeldverwerkingssysteem wordt gesteld is dat het binnen 50 msec de positie van de naad aan het robotbesturingssysteem moet kunnen leveren. In deze tijd van 50 msec is reeds de 20 msec inbegrepen die de camera nodig heeft om de beeldinformatie van één halfbeeld naar de uitgang van de camera te transporteren. 30 msec daarna moeten de coördinaten van de naadpunten naar het besturingssysteem zijn overgebracht (figuur 4.9).

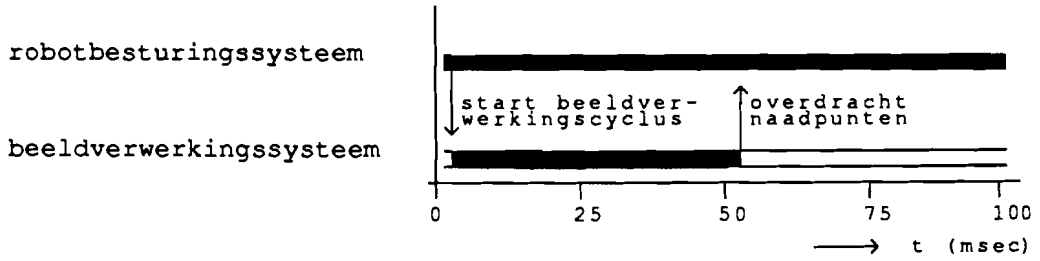
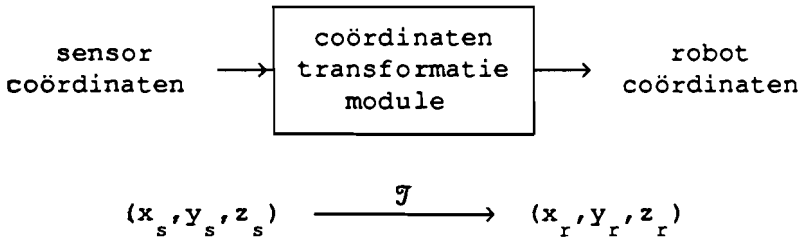


fig 4.9: Timing robotcyclus.
proces actief
proces non-actief

In hoofdstuk 5 wordt nader ingegaan op het ontwerp van de patroonherkenningsmodule.

4.5. De coördinatentransformatie

De patroonherkenningsmodule genereert een lijst met coördinaten van de gevonden naadpunten. De coördinaten zijn relatief t.o.v. het coördinatenstelsel van de sensor. De coördinatentransformatiemodule zet deze coördinaten om naar coördinaten relatief t.o.v. het coördinatenstelsel van de cartesische robot (figuur 4.10).



figuur 4.10: Coördinatentransformatiemodule.

Figuur 4.11 toont het sensorcoördinatenstelsel (het stelsel **S**) in het coördinatenstelsel van de robot (het stelsel **R**). De camera is verschoven t.o.v. de oorsprong van het robotcoördinatenstelsel langs de vector $\underline{Ox}_s$. De camera, die draaibaar is, bevindt zich in figuur 4.11 in de *referentie-stand.*
De assen x_s, y_s en z_s vallen samen met de referentieassen $x_{s,ref}$, $y_{s,ref}$ en $z_{s,ref}$

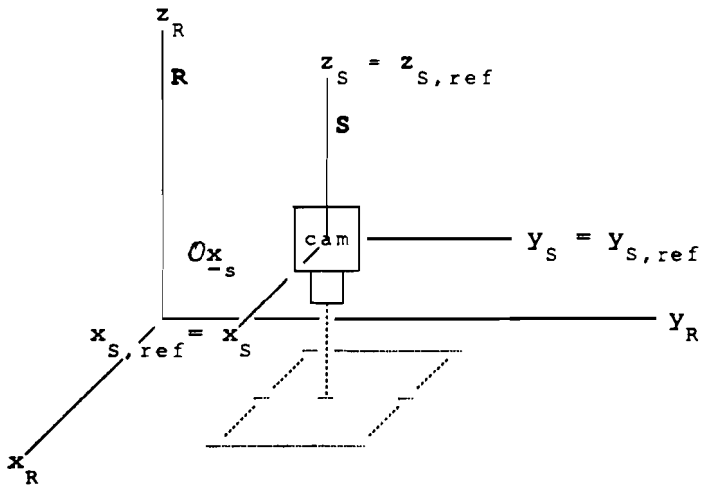


fig. 4.11: sensor- en robotcoördinatenstelsel (**S** en **R**).

In figuur 4.12 is de camera t.o.v. de referentie-assen gedraaid. Eerst

werd de camera gedraaid over een hoek θ om referentie-as $y_{S,ref}$ en daarna over een hoek ϕ om de referentie-as $z_{S,ref}$.

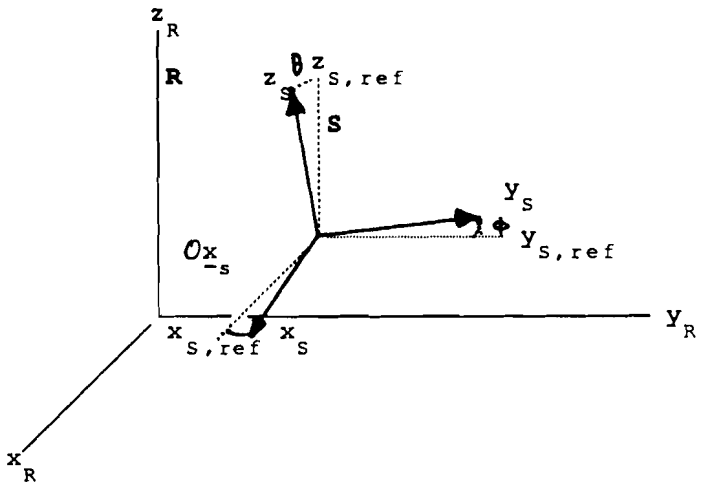


fig. 4.12: sensor- en robotcoördinatenstelsel (**S** en **R**).

Stel de patroonherkenningsmodule vindt het naadpunt P. Het punt P heeft de coördinaten (x_s, y_s, z_s) t.o.v. het stelsel **S**. De coördinatentransformatiemodule heeft als taak om de coördinaten (x_s, y_s, z_s) om te zetten naar de coördinaten (x_r, y_r, z_r) in het stelsel **R**.

De formule voor deze omzetting ziet er als volgt uit:

$$\begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} = \underline{Ox}_s + \mathbf{F}^{-1} \cdot \mathbf{R}_{y\text{-as},-\theta} \cdot \mathbf{R}_{z\text{-as},-\phi} \cdot \begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix}$$

$$= \begin{pmatrix} Ox_s \\ Oy_s \\ Oz_s \end{pmatrix} + \begin{pmatrix} \lambda^{-1} & 0 & 0 \\ 0 & \mu^{-1} & 0 \\ 0 & 0 & \nu^{-1} \end{pmatrix} \cdot \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix}$$

$$= \begin{pmatrix} \mathcal{O}x_s \\ \mathcal{O}y_s \\ \mathcal{O}z_s \end{pmatrix} + \begin{pmatrix} \lambda^{-1}\cdot\cos\theta\cdot\cos\phi & \lambda^{-1}\cdot\cos\theta\cdot\sin\phi & \lambda^{-1}\cdot\sin\theta \\ -\mu^{-1}\cdot\sin\phi & \mu^{-1}\cdot\cos\phi & 0 \\ -\nu^{-1}\cdot\sin\theta\cdot\cos\phi & -\nu^{-1}\cdot\sin\theta\cdot\sin\phi & \nu^{-1}\cdot\cos\theta \end{pmatrix} \cdot \begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix}$$

In de formule is de matrix $\mathbf{R}_{z\text{-as},-\phi}$ is een rotatie-operatie om de as $z_{s,ref}$ over een hoek $-\phi$. De matrix $\mathbf{R}_{y\text{-as},-\theta}$ is een rotatie-operatie om de as $y_{s,ref}$ over een hoek $-\theta$.
De schaling langs de assen van het stelsel **S** verschilt met die van het stelsel **R**. Bij de transformatie dient dus een correctie te worden uitgevoerd vanwege het verschil in schaling bij de twee stelsels. De elementen van de matrix **F** (dit zijn: λ, μ en ν) zijn de zogenaamde correctiefactoren.
De vector $\mathcal{O}\underline{x}_s$ wijst in het stelsel **R** de oorsprong van het stelsel **S** aan.

Bij het bepalen van de positie van de naad maakt de patroonherkenningsmodule gebruik van voorkennis over de positie van de lichtstrepen in het camerabeeld (zie hoofdstuk 5). Deze posities zijn opgeslagen in de host computer en deze zijn relatief t.o.v. het coördinatenstelsel van de robot. Voordat de patroonherkenningsmodule deze voorkennis kan benutten dienen deze posities te worden getransformeerd naar posities in sensorcoördinaten. De formule voor de transformatie van robotcoördinaten naar sensorcoördinaten ziet er als volgt uit:

$$\begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix} = \mathbf{R}_{z\text{-as},\phi}\cdot\mathbf{R}_{y\text{-as},\theta}\cdot\mathbf{F}\cdot\left(\begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} - \begin{pmatrix} \mathcal{O}x_s \\ \mathcal{O}y_s \\ \mathcal{O}z_s \end{pmatrix}\right)$$

$$= \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} \lambda & 0 & 0 \\ 0 & \mu & 0 \\ 0 & 0 & \nu \end{pmatrix} \cdot \left(\begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} - \begin{pmatrix} \mathcal{O}x_s \\ \mathcal{O}y_s \\ \mathcal{O}z_s \end{pmatrix}\right)$$

$$= \begin{pmatrix} \lambda\cdot\cos\phi\cdot\cos\theta & -\mu\cdot\sin\phi & -\nu\cdot\cos\phi\cdot\sin\theta \\ \lambda\cdot\sin\phi\cdot\cos\theta & \mu\cdot\cos\phi & -\nu\cdot\sin\phi\cdot\sin\theta \\ \lambda\cdot\sin\theta & 0 & \nu\cdot\cos\theta \end{pmatrix} \cdot \left(\begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} - \begin{pmatrix} \mathcal{O}x_s \\ \mathcal{O}y_s \\ \mathcal{O}z_s \end{pmatrix}\right)$$

In de transformatiematrices dient een aantal malen de sinus of cosinus van de hoeken θ of ϕ berekend te worden. Het berekenen van de sinus en de cosinus van een hoek kost relatief veel rekentijd in vergelijking met de rekentijd die nodig is voor een optelling of een vermenigvuldiging. Een gegeven is dat de camera slechts een beperkt aantal posities moet kunnen aannemen. Het is bijvoorbeeld voldoende wanneer de hoeken θ en ϕ de volgende waarden aan kunnen nemen:

$$\phi \in \{ 0^\circ, 30^\circ, 60^\circ, 90^\circ, 120^\circ, 150^\circ, 180^\circ, 210^\circ, 240^\circ, 270^\circ, 300^\circ, 330^\circ \}$$

$$\theta \in \{ 0^\circ, 30^\circ, 60^\circ, 90^\circ, 120^\circ, 150^\circ, 180^\circ \}$$

Door de sinussen en cosinussen van deze waarden op te slaan in een tabel, is men geen extra rekentijd meer kwijt aan het berekenen van de sinus en cosinus van de hoeken θ en ϕ.

Tabel 4.2: Tabel met sinussen en cosinussen

ψ	$\sin(\psi)$	$\cos(\psi)$
0	0	1
30	0.5	0.8660254
60	0.8660254	0.5
90	1	0
⋮	⋮	⋮
300	-0.8660254	0.5
330	-0.5	0.8660254

In eerste instantie gaan we er van uit van een naadvolgsysteem waarin de camera niet draaibaar is. De hoeken θ en ϕ hebben de vaste waarde, namelijk 0°. Met dit gegeven worden de transformatiematrices gelijk aan:

$$\begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} = \begin{pmatrix} \mathcal{O}x_s \\ \mathcal{O}y_s \\ \mathcal{O}z_s \end{pmatrix} + \begin{pmatrix} \lambda^{-1} & 0 & 0 \\ 0 & \mu^{-1} & 0 \\ 0 & 0 & \nu^{-1} \end{pmatrix} \cdot \begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix}$$

$$\begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix} = \begin{pmatrix} \lambda & 0 & 0 \\ 0 & \mu & 0 \\ 0 & 0 & \nu \end{pmatrix} \cdot \left(\begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} - \begin{pmatrix} \mathcal{O}x_s \\ \mathcal{O}y_s \\ \mathcal{O}z_s \end{pmatrix} \right)$$

Om rekentijd te besparen dienen de constanten λ^{-1}, μ^{-1} en ν^{-1} (elementen van de matrix $\mathbf{F}^{-1}$) ook vooraf berekend te worden.

5. Patroonherkenningsmodule

5.1. Inleiding

In hoofdstuk 4 zagen we dat het beeldverwerkingssysteem is opgebouwd uit vier modules. Dit zijn de sync-scheider, het pulsbreedtefilter en de modules waarin de patroonherkenning en de coördinatentransformatie worden uitgevoerd. In dit hoofdstuk zal het nader worden ingegaan op de functie van de patroonherkenningsmodule.

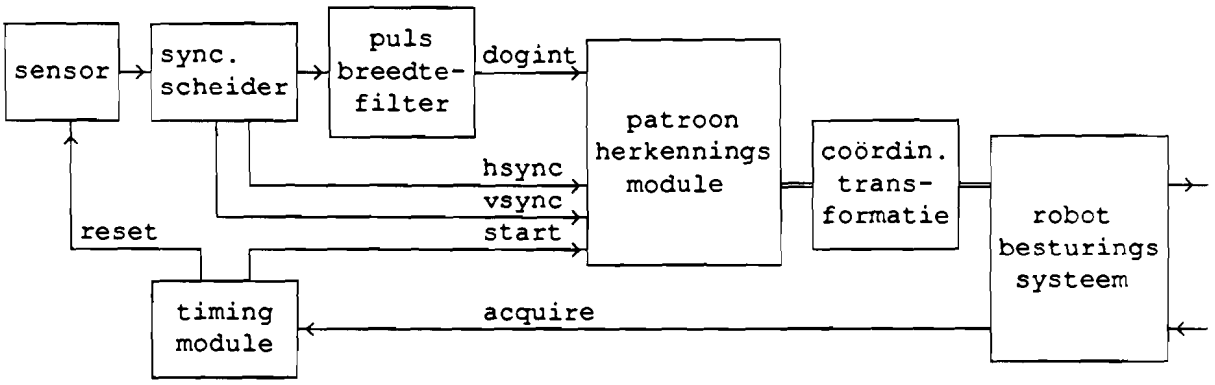


fig. 5.1: Het beeldverwerkingssyteem.

De patroonherkenningsmodule heeft een *input* en een *output*. De input is een camerabeeld met daarin het streepjespatroon, dat m.b.v. het pulsbreedtefilter is voorbewerkt. De output is een lijst coördinaten die de ligging van de gezochte naad beschrijft. Het blokschema van de patroonherkenningsmodule wordt getoond in figuur 5.1.

Telkens wanneer het pulsbreedtefilter een element van één van de lichtstrepen detecteert genereert de coördinatengenerator de corresponderende coördinaten. Het besturingssysteem heeft een bepaalde verwachting over de plaatsen in het camerabeeld waar de lichtrepen komen te liggen. Rondom deze plaatsen definiëert het beeldverwerkingssysteem zogenaamde *areas*. Over deze areas worden de streeppunten verdeeld. Bij elk area ontstaat een verzameling streeppunten die de

lichtstreep in dat area beschrijft. Eén zo'n verzameling streeppunten wordt *stripe* genoemd.

De *taak* van de patroonherkenningsmodule is om de positie van de naad te bepalen door de vorm van de lichtstrepen (stripes) te analyseren. Karakteristiek voor een overlapnaad is het sprongetje dwars op de richting van de lichtstreep.

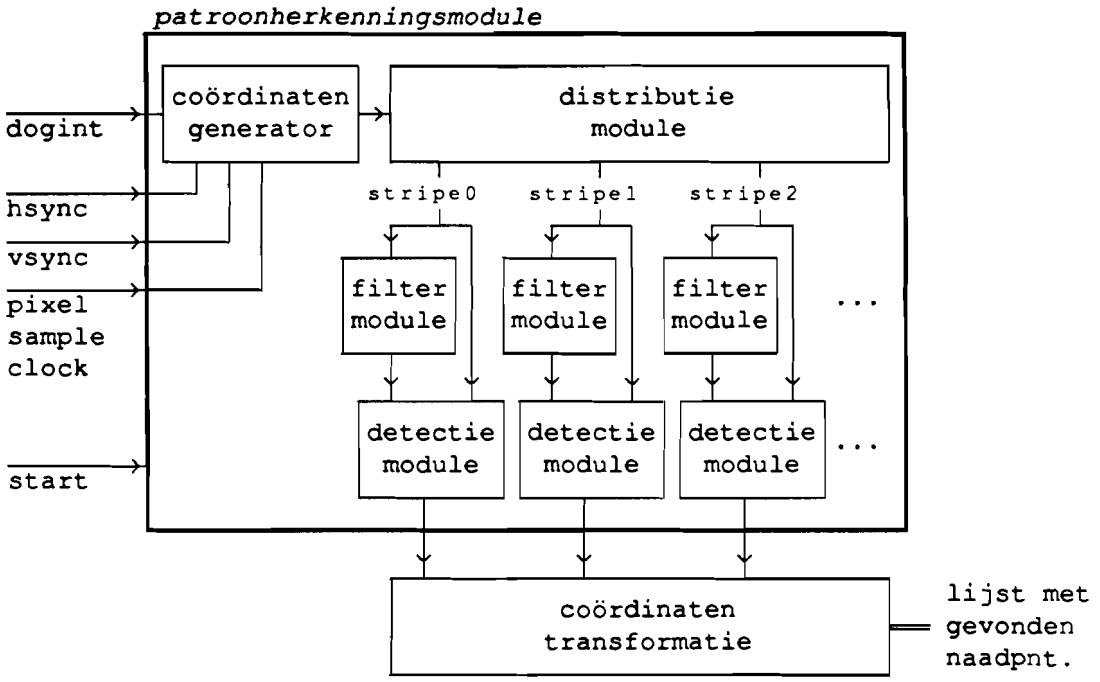


fig. 5.2: Patroonherkenningsmodule.

Een bruikbaar hulpmiddel bij de analyse van de stripes is een *edge detection filter.* Plotselinge overgangen in een signaal (edges) worden door dit soort filters versterkt. De responsie van het edge detection filter is afhankelijk van de grootte van de sprong. Wanneer in de stripe een bepaald sprongetje optreedt dan beslist de detectiemodule of dit correspondeert met een naad die valt binnen de categorie gezochte naden.

5.2. De ADSP-2100 digitale signaalprocessor

De coördinatengenerator is geheel in digitale hardware uitgevoerd. De andere modules zijn uitgevoerd op een digitale signaalprocessor in software. Het voordeel om bepaalde modules uit te voeren in software is dat deze modules eenvoudig kunnen worden gewijzigd. Dit gaat bij hardware meestal minder eenvoudig.
De motivatie om te kiezen voor een digitale signaalprocessor is het feit dat de *stripe datasets* zullen worden bewerkt m.b.v. een digitaal *edge enhancement* filter. De architectuur van digitale signaalprocessoren maakt hen zeer geschikt om ze te gebruiken als bouwsteen voor digitale filters.

Gekozen werd om de patroonherkenningsmodule uit te voeren op de ADSP-2100 digitale signaal processor van Analog Devices [1]. De snelheid waarmee de ADSP-2100 een filterbewerking (edge enhancement) uitvoert is redelijk in vergelijking met andere digitale processoren (figuur 5.3, bron: *"EDN's DSP Benchmarks"* [11]).

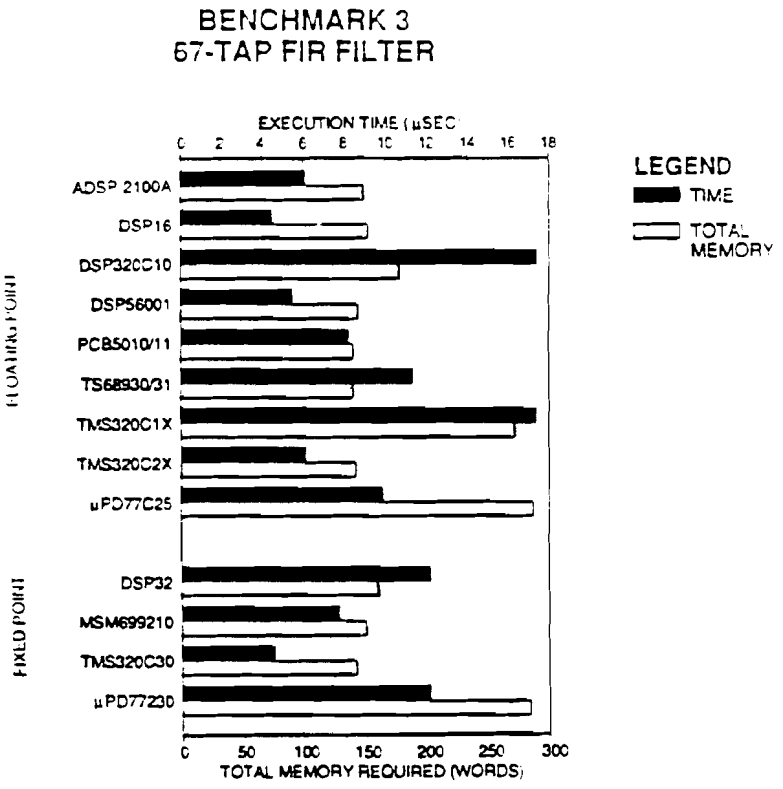


fig. 5.3: Workbench: digitaal filter.

Analog Devices levert een softwarepakket, dat dient om de gebruiker te als helpen bij het ontwikkelen van programmatuur voor de ADSP-2100. Dit softwarepakket noemt men de *Cross-software* (zie [2]). Het softwarepakket wordt geïnstalleerd op een IBM compatible PC en het bevat de volgende modules:

- de ADSP-2100 System builder
- de ADSP-2100 C-compiler
- de ADSP-2100 Assembler
- de ADSP-2100 Linker
- de ADSP-2100 Simulator
- de ADSP-2100 PROM-splitter

De eerste module is de System builder. Voordat kan worden begonnen met het ontwikkelen van programmatuur dient eerst de hardware configuratie van de ADSP-2100 m.b.v. de System builder te worden vastgelegd.
De volgende twee modules zijn de C-compiler en de Assembler. Het grootste deel van de programmatuur wordt ontwikkeld in de hogere programmeertaal C [9]. Bepaalde programmatuur echter, zoals interrupt routines en zeer tijdkritische programma's, wordt geprogrammeerd in de ADSP-2100 assembly code. M.b.v. de C-compiler en de assembler wordt de source code omgezet in objectfiles.
De linker-module genereert uit deze objectfiles een lijst met machine-instructies voor de ADSP-2100.
Voordat deze instructies in het geheugen van de ADSP-2100 worden geplaatst, kan men de correctheid van de ontwikkelde programmatuur testen met de ADSP-2100 simulator.
M.b.v. de PROM Splitter kunnen de machine-instructies worden overgebracht naar een PROM of een EPROM.

Typische eigenschappen van de ADSP-2100 zijn:

- Elke instructie wordt in één processorcyclus uitgevoerd. Eén instructiecyclus duurt minimaal 80 nsec. De processorsnelheid is dus maximaal 12.5 Mips (Mega instructions per second).
- Het geheugen is gesplitst in een gedeelte waar het programma wordt opgeslagen (program memory, PM) en een gedeelte waar variabelen etc. worden opgeslagen (data memory, DM). Het programmageheugen en het datageheugen hebben elk een eigen adresbus en eigen databus (zie figuur 5.4)

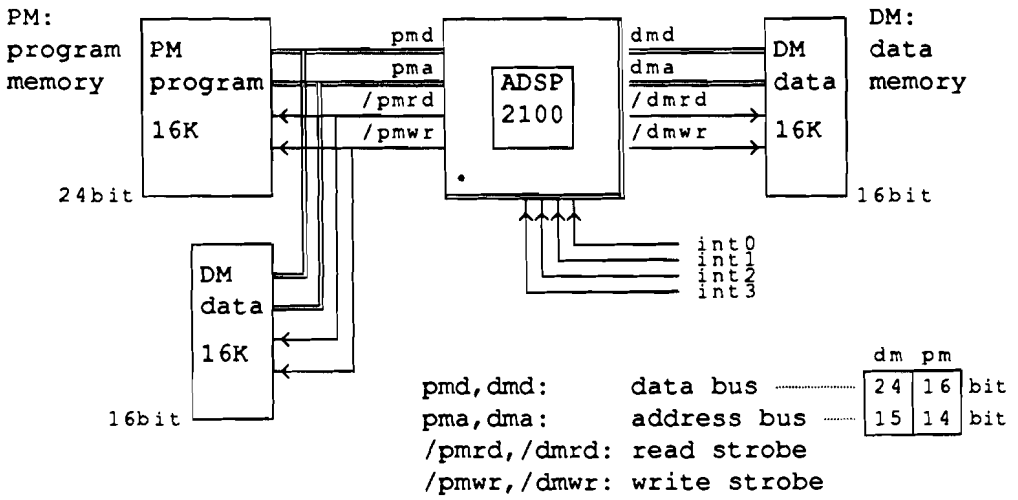


fig. 5.4: Busarchitectuur ADSP-2100.

- De digitale signaalprocessor heeft een cache geheugen met een grootte van 16 instructies. Wanneer een programmalus wordt uitgevoerd met maximaal 16 instructies, dan behoeven de instructies niet steeds opnieuw uit het programmageheugen te worden opgehaald. De snelheid waarmee de programmalus wordt doorlopen neemt daardoor toe.
- Het uitvoeren van een vermenigvuldiging duurt één processorcyclus.
- Een aantal instructies kan parallel worden uitgevoerd.
  Een voorbeeld is de volgende *multi-function* instructie:

  ```
  mr = mr + mx0*my0, mx0 = dm(i0,m0), my0 = pm(i4,m4);
  ```

  Binnen één instructiecylus worden tegelijkertijd een vermenigvuldiging (mx0*my0), een optelling (mr = mr + ...) en twee *data move* instructies (dm(...) en pm(...)) uitgevoerd.
  In paragraaf zal de bovenstaande multi-function instructie in detail worden besproken.
- Bij programmalussen hebben we te maken met een bepaalde overhead. Zo moet tijdens de uitvoer van de programmalus een teller worden opgehoogd en moet de telstand bijvoorbeeld worden vergeleken met een maximum waarde.
  Door gebruikmaking van de zogenaamde *do...untill* instructie wordt deze overhead geëlimineerd. Extra hardware in de ADSP-2100 zorgt ervoor dat de programmalus het juiste aantal malen wordt doorlopen.
- Door de aanwezigheid van schaduwregisters is een zeer snelle interruptafhandeling mogelijk. Wanneer een interruptroutine

moet worden afgehandeld, kan de inhoud van de dataregisters worden veilig gesteld door in de routine de schaduwregisters te gebruiken.
- Er is intern een interrupt controller aanwezig voor de interrupt afhandeling van de interruptaanvragen op de vier interruptingangen.

Analog Devices heeft een zogenaamd *evaluation board* ontworpen waarmee realtime de software kan worden getest die m.b.v. de cross-software is ontwikkeld. Dit evaluation board bevat:

- Een ADSP-2100 digitale signaalprocessor
- *Sockets* voor:
  - een 16 Kbyte programma geheugen (PM, programma gedeelte)
  - een 16 Kbyte programma geheugen (PM, data gedeelte)
  - een 16 Kbyte data geheugen (DM)
- Een aantal I/O faciliteiten zoals een A/D omzetter en een D/A omzetter.
- Een connector die het mogelijk maakt om eigen applicaties op de datageheugen-busstructuur van de processor aan te sluiten.

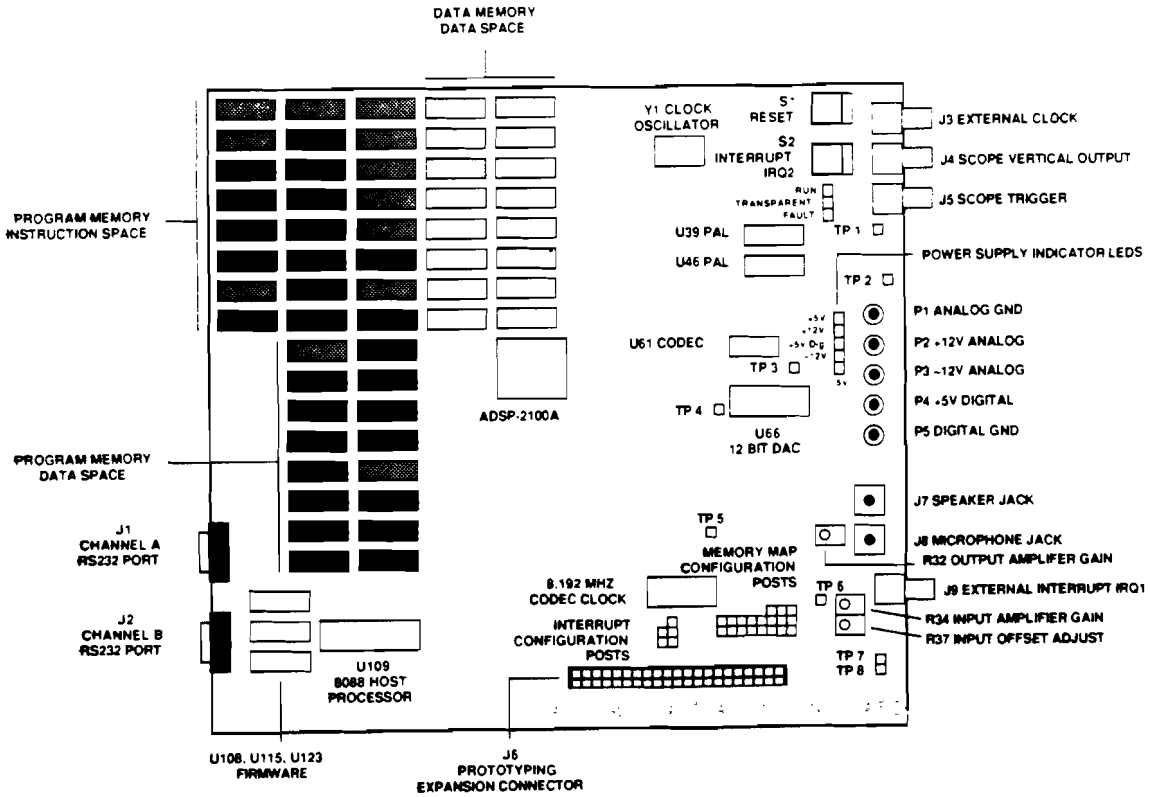


fig. 5.5: Het ADSP-2100 evaluation board.

We ontwikkelen de beeldverwerkingsoftware in programmeertaal C. Deze software wordt gecompileerd m.b.v. de op een IBM PC geïnstalleerde ADSP-2100 cross-software.
Om de door de cross-software gegenereerde machinecode te testen op het ADSP-2100 evaluation board, dient deze code te worden overgebracht naar het programmageheugen van de signaalprocessor. Dit kan geschieden via een seriële datalijn, die de IBM PC met het evaluation board verbindt. Daarbij dient men gebruik te maken van een *terminal emulation software* zoals VTERM, PROCOMM of SMARTTERM (in ons geval VTERM, zie appendix I)

De implementatie van de coördinatengenerator kon worden gerealiseerd m.b.v. een stukje digitale electronica. De distributiemodule, de DOG-filters, de patroonherkenning en de coördinatentransformatie werden uitgevoerd in software op de ADSP-2100. In figuur 5.6 is te zien hoe de coördinatengenerator werd ingepast in de bus-architectuur van de ADSP-2100. De overdracht van de coördinaten, die het streepjespatroon beschrijven in het videobeeld, naar digitale signaalprocessor gaat via twee dataregisters. Deze dataregisters liggen in de adresruimte van het data geheugen (DM).

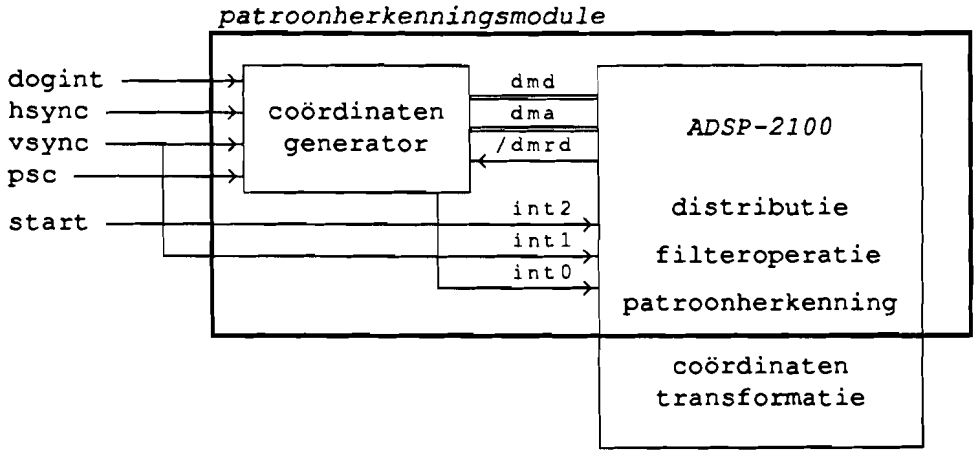


fig. 5.6: Patroonherkenningsmodule.

In paragraaf 5.3 zal worden getoond hoe m.b.v. digitale hardware de coördinatengenerator is opgebouwd. De distributie-, de filter-, de detectie- en de transformatiemodule zijn op de ADSP-2100 geïmplementeerd in software en deze worden besproken in de paragrafen 5.4 tot en met 5.6.

5.3. De coördinatengenerator

In figuur 5.7 is de coördinatengenerator weergegeven als een *black box*.

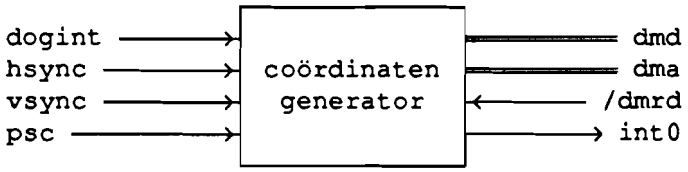


fig. 5.7: De coördinatengenerator.
(psc = pixel sample clock)

De taak van de coördinatengenerator is om bij elk pulsje, dat door het pulsbreedte filter wordt aangeboden, de corresponderende beeldcoördinaten te bepalen. Figuur 5.8 toont hoe het camerabeeld is opgesplitst in pixels. De horizontale coördinaat noemen we x-coördinaat en de verticale coördinaat zal y-coördinaat worden genoemd.

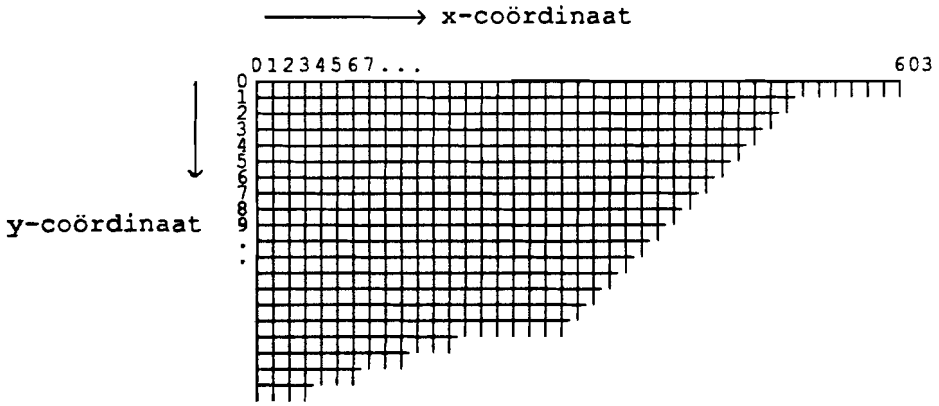


fig. 5.8: Beeldcoördinaten (pixels).

De *pixel sample clock* deelt de beeldlijnen op in pixels. De periodetijd van deze pixel sample clock correspondeert met de breedte van één pixel. De x-coördinaat van een pixel in het videosignaal wordt bepaald door het aantal perioden van de pixel sample clock vanaf het tijdstip dat de horizontale synchronisatie puls werd ontvangen.
De y-coördinaat wordt bepaald door het aantal horizontale synchronisatiepulsen vanaf het moment dat de verticale synchronisatie werd gedetecteerd. Figuur 5.9 toont een eenvoudige digitale schakeling voor het genereren van deze beeldcoördinaten.

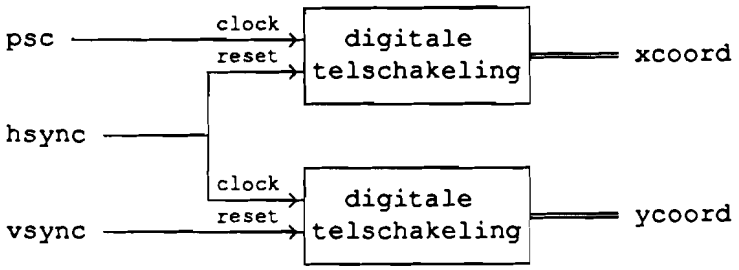


fig. 5.9: Coördinatengeneratie.

Het complete blokschema van de coördinatengenerator is weergegeven in figuur 5.10. Op het moment dat het pulsbreedte filter een dogint pulsje genereert wordt de inhoud van de telschakelingen geladen in twee data registers, genaamd XREG en YREG. Via de uitgang dogint* laat de coördinatengenerator aan de digitale processor weten dat er een nieuw element van het streepjespatroon is gedetecteerd en dat coördinaten staan opgeslagen in de twee genoemde data registers. M.b.v. twee leescycli kunnen de x- en y-coördinaten worden uitgelezen. De registers XREG en YREG zijn geadresseerd in het data geheugen van de ADSP-2100 op de geheugenplaatsen h#3ffe resp. h#3fff (appendix C)

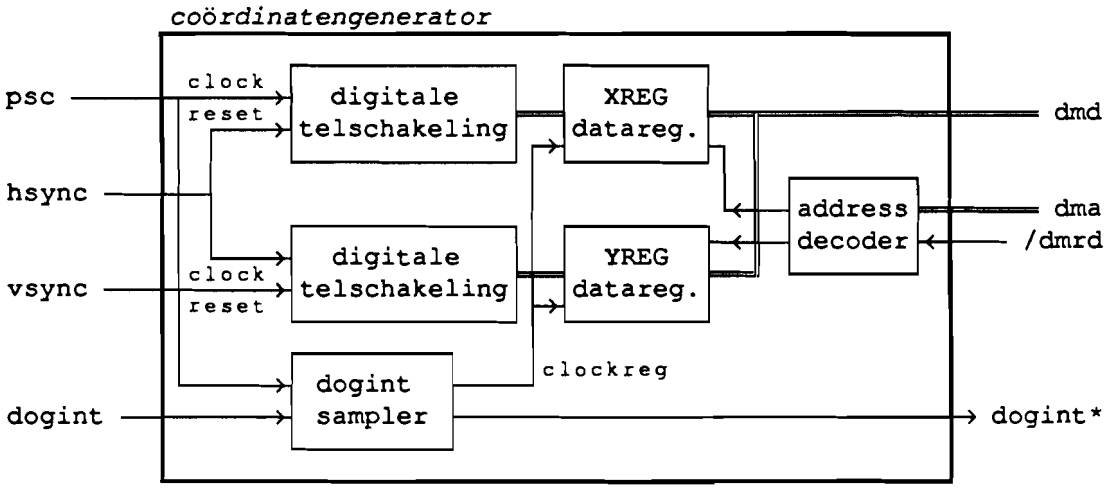


fig. 5.10: Blokschema coördinatengenerator.

De pixel sample clock is aangesloten op de *clock*-ingang van de digitale telschakeling waarmee de x-coördinaat van een pixel wordt bepaald. Op het moment dat er op de clock-ingang een opgaande flank verschijnt wordt de telstand met één verhoogd. De uitgang van de telschakeling is op dat moment niet betrouwbaar. De *dogint sampler* zorgt ervoor dat de telstanden pas in de dataregisters worden ingeladen wanneer in het psc-signaal een neergaande flank optreedt. Hiertoe genereert de dogint sampler het signaal: *clockreg*.

Telkens wanneer er een pulsje verschijnt in het dogint-signaal, dient de signaalprocessor de inhoud van de registers XREG en YREG in te lezen. De processor kan voor deze taak geactiveerd worden d.m.v. een interrupt op één van de interruptingangen.
De interruptingangen van de ADSP-2100 zijn laagactief en worden bemonsterd op het eind van de zevende periode van de instructiecyclus (zie figuur 5.11). Het gevolg hiervan is dat het interrupt signaal gedurende een periode van minstens de instructiecyclustijd laag moet zijn. Echter het pulsje op het dogint-signaal heeft een pulsduur van 45 nsec terwijl de cyclustijd gelijk is aan minimaal 80 nsec en maximaal 125 nsec (dit is afhankelijk van de gebruikte clock generator). De dogint-sampler verlengt het pulsje van 45 nsec tot een pulsje van minimaal 180 nsec (dit is twee maal de periodetijd van de pixel sample clock) en maximaal 270 nsec (drie maal de periodetijd van de pixel sample clock). De uitgang van de dogint-sampler met de verlengde pulsjes is genaamd dogint*.

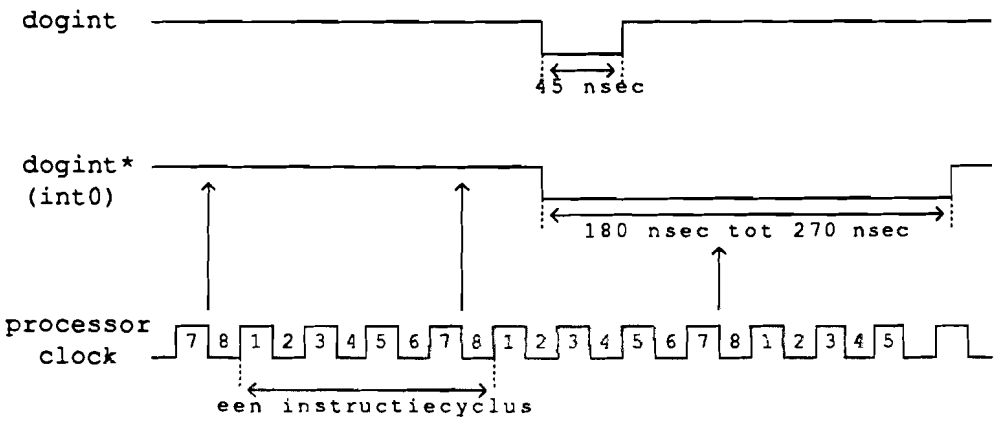


fig. 5.11: Bemonstering interruptingang.

De elektrische schema's van de coördinatengenerator zijn opgenomen in appendix E.

5.4. De distributiemodule

De distributiemodule heeft drie taken. De eerste taak is om de coördinaten van de streeppunten op te halen uit de registers XREG en YREG van de coördinatengenerator en deze op te slaan in een tabel. De tweede taak is om de streeppunten te verdelen over een aantal areas waarin het camerabeeld is opgedeeld. Per area ontstaat een verzameling beeldpunten die de lichtstreep in het betreffende area beschrijft. De derde taak is om de ongedefiniëerde stukken die in de verzemling streeppunten kunnen voorkomen op een verstandige manier te reconstrueren.

Timing:

Figuur 5.12 toont de signalen die op de interruptingangen van de ADSP-2100 zijn aangesloten.

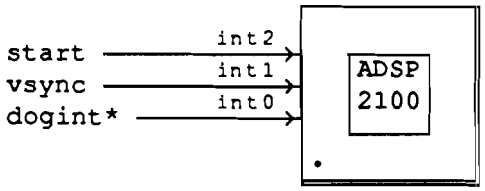


fig. 5.12: Interruptingangen ADSP-2100.

De patroonherkenningsmodule mag pas een nieuwe beeldverwerkingscyclus beginnen wanneer het via de timing module (zie figuur 5.1) van het besturingssysteem een startpuls heeft ontvangen. Wanneer deze puls is ontvangen dan leest de module de beeldinformatie in van het eerstvolgende halfbeeld. Het begin en het eind van dit halfbeeld wordt aangegeven d.m.v. de verticale synchronisatie.
In figuur 5.13 is aangegeven hoe de gebeurtenissen elkaar opvolgen. Het detecteren van een startpuls heeft als gevolg dat de signaalprocessor overgaat naar de toestand waarin hij de eerste verticale synchronisatie afwacht. Wanneer deze is ontvangen komt de signaalprocessor in een toestand terecht waarin hij de dogint* pulsjes verwerkt en tegelijkertijd de tweede verticale synchronisatie afwacht. Nadat die ook is gedetecteerd gaat de processor over naar de toestand waarin de ontvangen beeldinformatie verwerkt wordt. Is de verwerking voltooid dan komt de processor weer in de begintoestand waarin een nieuwe startpuls van het besturingssysteem wordt afgewacht.

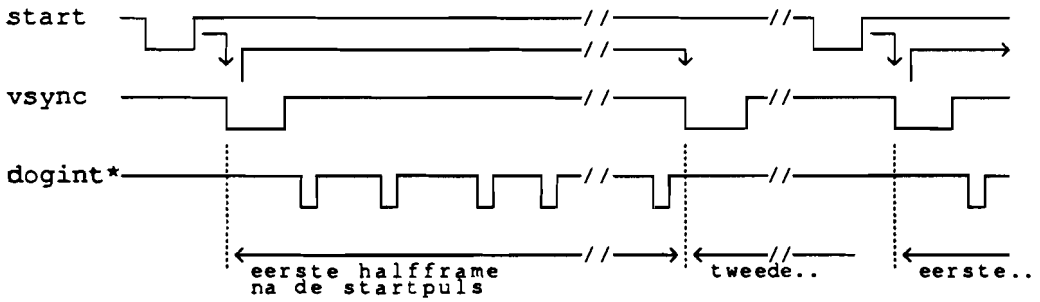


fig. 5.13: Timingdiagram.

Elke toestand (of beter fase) is genummerd. Figuur 5.14 geeft aan hoe de interruptingangen in de verschillende fases zijn gemaskeerd.

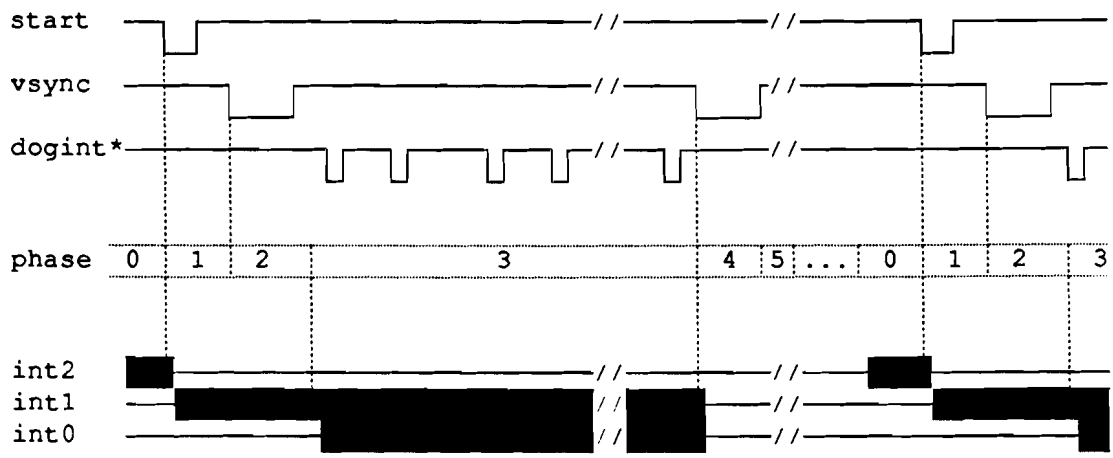


fig. 5.14: Timingsdiagram interruptingangen:
– : interruptingang gemaskeerd
▮ : interruptingang niet gemaskeerd

De interruptingangen zijn laagactief. In de vorige paragraaf werd reeds vermeld dat de ADSP-2100 zijn interruptingangen bemonstert op het einde van de zevende periode van de instructiecyclus. Elk van deze interruptingangen kan worden geprogrammeerd in de *level sensitive mode* of in de *edge sensitive mode*.

In de level sensitive mode reageert de processor wanneer het niveau van de interruptingang in de zevende periode van de instructiecyclus laag is. Indien het niveau nog steeds laag is, na uitvoer de interrupt serviceroutine, dan reageert de processor opnieuw.
Is de interruptingang in de edge sensitive mode geprogrammeerd, dan reageert de processor wanneer het niveau in de zevende periode van de instructiecyclus laag is, terwijl dit in de vorige cyclus nog hoog was. Is het niveau na uitvoer van de interrupt serviceroutine nog steeds laag dan reageert de processor niet.

In eerste instantie waren in ons geval de interruptingangen geprogrammeerd in de edge sensitive mode. Immers in dat geval ontstaan er niet direct problemen wanneer een ingang te lang actief blijft. Echter bij tests blijkt dat de processor in 50% van de gevallen direct van fase 2 naar fase 3 overgaat zonder de eerste synchronisatie puls op ingang int1 af te wachten. Mogelijk reageert de processor op een storing die de synchronisatie puls net vóór is. Het tijdstip, waarop de processor een interrupt detecteert, valt samen met het tijdstip waarop het

startsignaal een overgang maakt van laag naar hoog. De storing op de ingang int1 ontstaat waarschijnlijk als gevolg van capacitieve overspraak (zie verder § 6.3).

Dit probleem kan verholpen worden door de interruptingangen in level sensitive mode te programmeren. De puls op interrupt ingang wordt pas geaccepteerd wanneer deze ingang een aantal malen (bijvoorbeeld 25) de uitvoer van de interrupt serviceroutine heeft veroorzaakt. Wanneer het een stoorpulsje betreft met een zeer korte pulsduur, dan zal dit stoorpulsje waarschijnlijk slechts één maal de uitvoering van de interrupt serviceroutine veroorzaken. Het stoorpulsje wordt dan niet als interrupt geaccepteerd. De pulsduur van de verticale synchronisatie is in ieder geval groot genoeg om wel geaccepteerd te worden.

In fase 2 wordt een aantal variabelen geïnitialiseerd en daarna begint fase 3. In deze fase is interruptingang int0 gevoelig gemaakt voor pulsjes op het dogint*-signaal. Een pulsje op het dogint*-signaal heeft als gevolg dat de interrupt serviceroutine wordt uitgevoerd, die de coördinaten van het streeppunt ophaalt uit de dataregisters XREG en YREG en deze opslaat in de zogenaamde *pixel-tabel* (figuur 5.16). Het paar coördinaten komt op de plaats te staan waar de *pixel pointer (pixptr)* naar wijst. Nadat daar een nieuw paar is geplaatst, wijst de pixel pointer naar de eerste volgende lege element van de tabel.

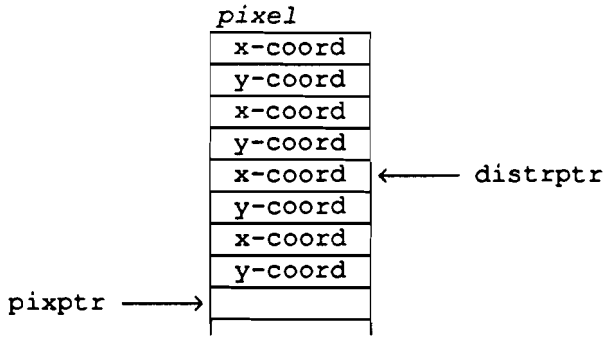


fig. 5.16: Pixel-tabel met pixel- en distributiepointers.

In de periodes, dat de serviceroutine (behorende bij interruptingang int0) niet wordt uitgevoerd, is de processor bezig met het verwerken van de streeppunten die reeds in de pixel-tabel zijn opgeslagen. De zogenaamde distributie pointer wijst naar het streeppunt dat op dat moment wordt verwerkt.
De taak van de distributiemodule is om de ontvangen coördinaten te verdelen over een aantal gebieden waarin het camerabeeld is opgedeeld. Deze gebieden worden getoond in figuur 5.17. De grenzen van deze gebieden worden bepaald door de x-coördinaten die staan opgeslagen in

het array *area_bound[]*.
Deze grenzen worden berekend met de formule:

$$area\_bound[i] = \frac{stripe\_x[i] + stripe\_x[i+1]}{2}$$

In deze formule is *stripe_x[i]* gelijk aan de verwachte positie (x-coördinaat) van de i^e lichtstreep in het camerabeeld. De verwachting van deze positie dient vanuit de host computer (waar deze als voorkennis is opgeslagen) in fase 2 naar de patroonherkenningsmodule te worden overgebracht.

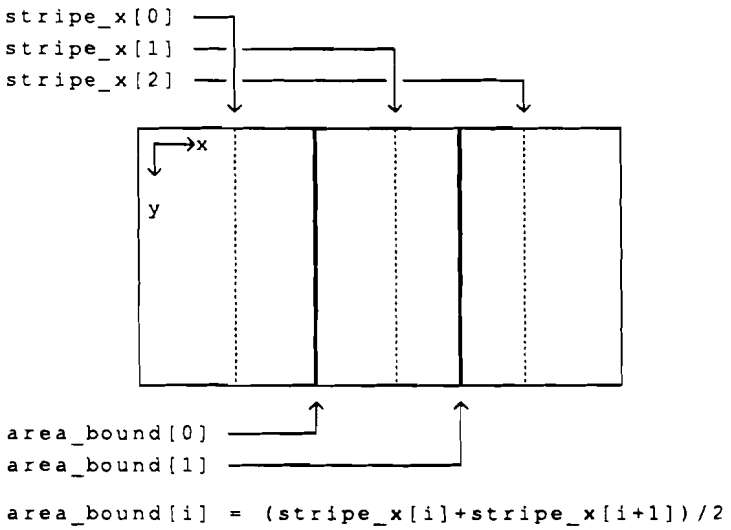


fig. 5.17: Camerabeeld met daarin:
: de plaatsen waar het besturingssysteem de lichtstrepen verwacht,
| : de area-grenzen.

Distributie

De distributiemodule verdeelt de coördinaten over de verschillende areas. Per area worden de coördinaten opgeslagen in de *stripe* datastructuur. Een stripe is een twee dimensionaal array waarin de x-coördinaten van de gevonden streeppunten worden opgeslagen. Deze x-coördinaat wordt opgeslagen in het array-element met als eerste index de area waartoe het streeppunt behoort en als tweede index de y-coördinaat van het streeppunt.

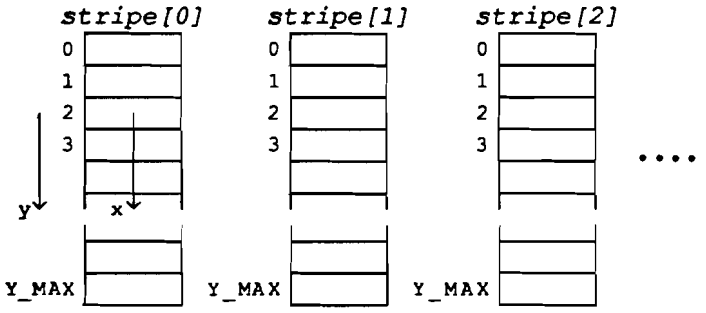


fig. 5.18: De stripe datastructuur.

Bij het verdelen van de streeppunten over de verschillende areas gedraagt de distributiemodule zich *conservatief*.
Dit betekent het volgende:
Stel de x-coördinaat van een nieuw streeppunt wijkt sterk af van de x-coördinaat van het streeppunt dat het laatst aan de stripe datastructuur werd toegevoegd - d.w.z. het verschil in de x-çoördinaten is groter dan de waarde opgeslagen in de variabele *displacement*. In dit geval wordt dit nieuwe streeppunt niet meteen in de stripe datastructuur opgeslagen, immers de kans is groot dat het nieuwe streeppunt een storing is. Pas wanneer blijkt dat de volgende x-coördinaten van de streeppunten ook sterk afwijken dan hebben we te maken een nieuwe *trend* en worden de streeppunten in de stripe datastructuur opgeslagen. Er is sprake van een nieuwe trend wanneer het aantal pixels dat sterk afwijkt van de oude trend groter is de waarde opgeslagen in de variabele *smooth_factor*.

Reconstructie

In figuur 5.2 zagen we dat de distributiemodule de verzamelingen streeppunten aanbiedt aan de filtermodules. De filtermodules voeren elk een edge enhancement operatie uit op de inputdata om de plaats waar de naad optreedt te accentueren (zie § 5.5).
Een probleem is dat in de lichtstreep meestal onderbrekingen optreden, met het gevolg dat een aantal elementen van de stripe-arrays niet gedefiniëerd zijn. Voordat de filtermodules op de stripe-arrays een filterbewerking uit kan voeren, dienen de ongedefiniëerde stukken te worden ingevuld (oftwel gereconstrueerd).
Hoe dit op een verstandige manier kan gebeuren wordt hieronder beschreven.

Een werkstuk wordt belicht door een projector. Figuur 5.19 toont het

streepjespatroon van het voorbewerkte camerabeeld. In deze figuur is de projector gepositioneerd aan de linkerkant van het camerabeeld. Hoe verder de strepen links in het camerabeeld liggen des te kleiner is de afstand tussen de camera en het werkstuk.

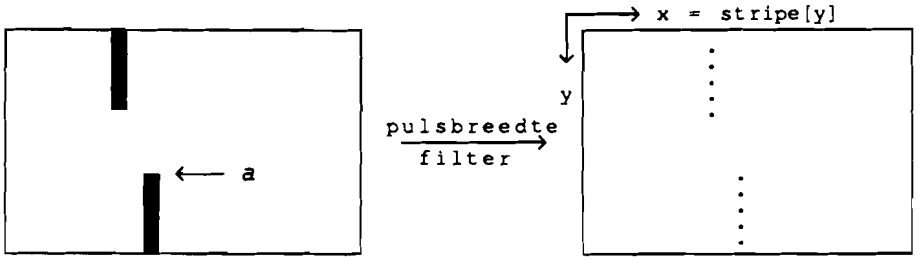


fig 5.19: Lichtstreep en stripe.

Het metaal in de buurt van de naad is enigszins afgerond. De lichtintensiteit van de lichtstrepen is daar zodanig dat ze niet door het pulsbreedte filter worden gedetecteerd. De opening tussen de twee metalen platen bevindt zich in het camerabeeld op het punt dat met de letter *a* is gemarkeerd.

Wanneer de distributiemodule in het camerabeeld punten toevoegt zoals getoond in figuur 5.20 dan komt de overgang in de stripe precies te liggen op de plaats waar de naad ligt.

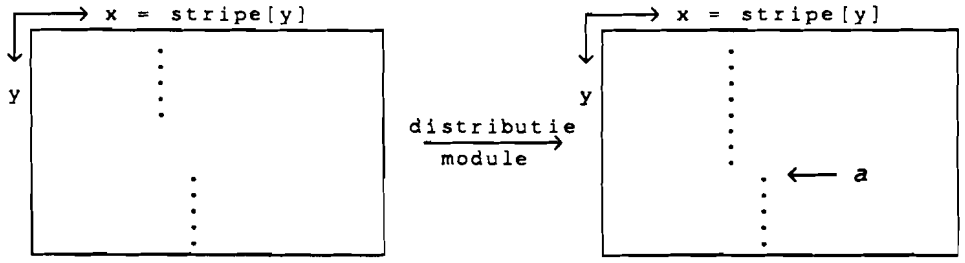


fig. 5.20: Stripe (type a) omzetten in een functie.

Het toevoegen van deze punten kan worden uitgevoerd m.b.v. een een-

voudig (lees: snel) algoritme. Wanneer op één van de beeldlijnen geen streeppunt voorkomt dan dient daar alsnog een streeppunt te worden geplaatst en wel zodanig dat de x-coördinaat van dat streeppunt gelijk is aan de x-coördinaat van het streeppunt op de beeldlijn erboven.

Bij het voorbeeld in figuur 5.21 ligt het hoger gelegen gedeelte van het werkstuk onder in het camerabeeld. De positie van de opening tussen de twee metalen platen is aangeduid met de letter *b*.

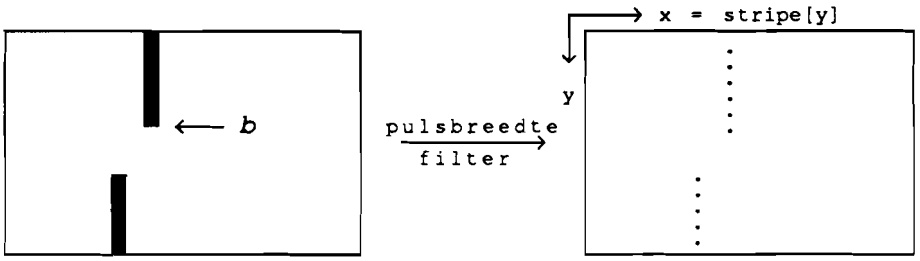


fig 5.21: Lichtstreep en stripe.

Bij dit voorbeeld dienen punten toegevoegd te worden zoals getoond in figuur 5.22 om de overgang op de positie van de naad te krijgen.

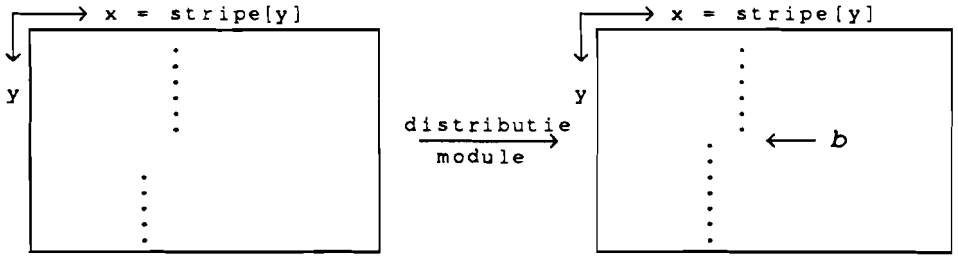


fig. 5.22: Stripe (type b) omzetten in een functie.

De naad in figuur 5.21 noemen we een overlapnaad van het type *a*. De naad in figuur 5.22 behoort tot het type *b*. Bij het omzetten van de stripe moet de distributiemodule voorkennis hebben over het type naad. Deze voorkennis moet geleverd worden door het besturingssysteem.

5.5. De filtermodule

De stripe datasets worden aangeboden aan een discreet lineair filter. Het uitgangssignaal u(n) van het filter is de convolutie van het ingangssignaal f(n) met de impulsresponsie g(n):

$$u(n) = g(n) * f(n) = \sum_{m=0}^{M} g(m) \cdot f(n+m) \qquad (n \in \{1,2,..N\})$$

fig 5.23: Discreet lineair filter.

Figuur 5.24 toont een voorbeeld van een stripe met daarin een plotselinge overgang en een stripe met een meer geleidelijke overgang. Een *edge enhancement* filter kan worden gebruikt om de overgangen in de stripes te *versterken*.

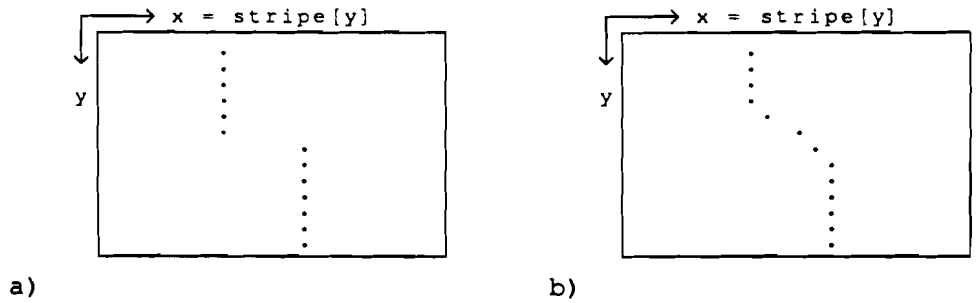


fig. 5.24: a) Stripe met plotselinge overgang
b) Stripe met geleidelijke overgang

Een voorbeeld van een edge enhancement filter is het DOG-filter (zie § 4.3). Figuur 5.25 toont de impulsresponsie en de stapresponsie van

een discreet DOG-filter met de filtercoëfficiënten:

-2 -6 -6 -2 0 2 6 8 8 6 2 0 -2 -6 -6 -2

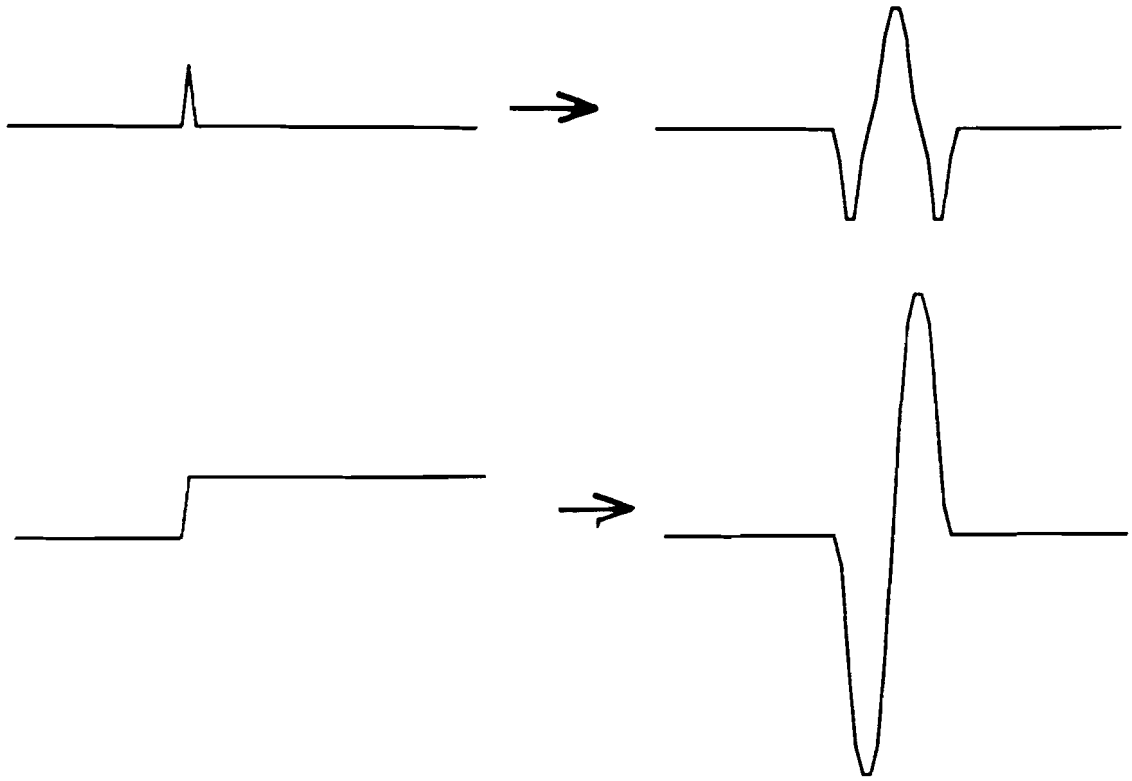


fig 5.25: Impuls- en stapresponsie van een DOG-filter met 16 filter coëfficiënten.

De responsies op meer geleidelijke overgangen worden getoond in figuur 5.26.

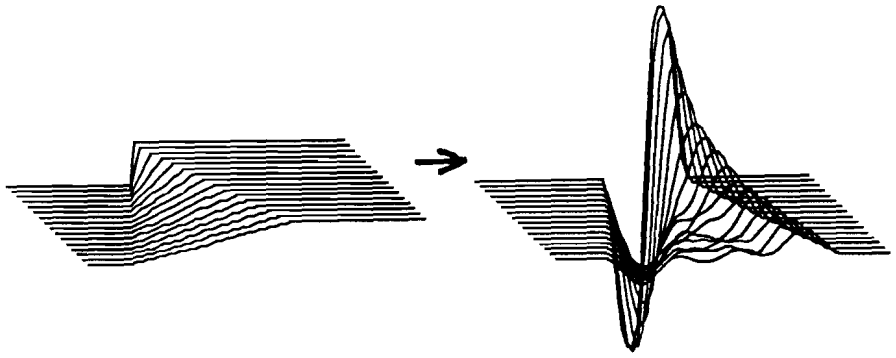


fig 5.26: DOG-filter responsies op een signaal met meer geleidelijke overgangen.

We definiëren de *geleidelijkheid* van een overgang in een stripe als de breedte van het schuine deel in het ingangssignaal uitgedrukt in pixels. In figuur 5.26 loopt de geleidelijkheid op van 0 tot 15. De responsie van het DOG-filter is relatief groot wanneer de geleidelijkheid van de overgang kleiner blijft dan de breedte van het postieve deel van de impulsresponsie.
Een meer geleidelijke overgang kan alleen versterkt worden met een nog *breder* DOG-filter, bijvoorbeeld een filter met filtercoëfficiënten:

-1-2-3-3-3-3-2-1 0 1 1 2 3 3 4 4 4 4 3 3 2 1 1 0-1-2-3-3-3-3-2-1

De responsie van dit filter op een stapvormige overgang en op meer geleidelijke overgangen wordt getoond in figuur 5.27.

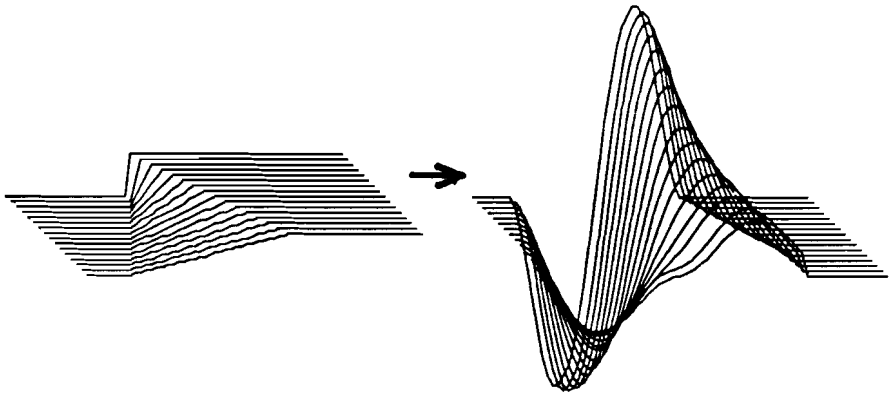


fig 5.27: Responsies van een DOG-filter met 32 filter coëfficiënten.

De input voor het DOG-filter staat opgeslagen in de stripe datastructuur. De output wordt opgeslagen in de convol datastructuur (figuur 5.28).

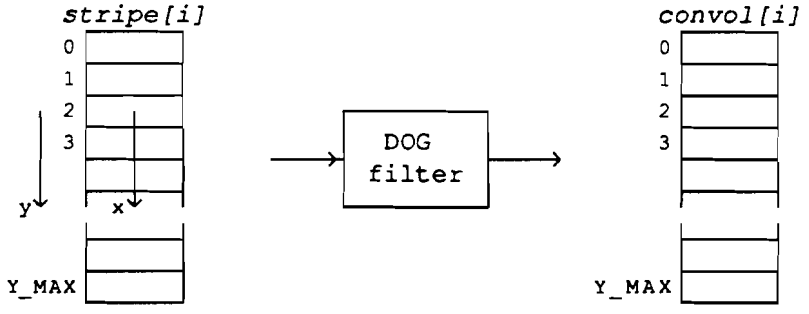


fig. 5.28: DOG-filter: *input*: stripe[i]
output: convol[i]

Het stukje programma voor het uitvoeren van de filterbewerking is niet geschreven in de hogere programmeertaal C, maar in de assemlytaal van de ADSP-2100. Dit is gedaan om optimaal gebruik te kunnen maken van de faciliteiten die de ADSP-2100 biedt om een filterbewerking in een zo kort mogelijke tijd uit te voeren.

Het i^e element van de DOG-filter responsie (dit is: *convol[area][i]*) op het *ingangssignaal stripe[area]* wordt m.b.v. onderstaande formule berekend:

$$convol[area][i] = \sum_{j=0}^{15} dog\_kernel[j] \cdot stripe[area][i+j]$$

In de assembly-code van de ADSP-2100 komt deze formule er als volgt uit te zien:

```
        {stripe_ptr = &stripe[area][i];}
        {convol_ptr = &convol[area][i];}
        {m1 = 1 en m5 = 1}

 1       i1=dm(stripe_ptr_);
 2       i5=^dog_kernel;
 3       mr=0, mx0=dm(i1,m1), my0=pm(i5,m5);
 4       cntr=15
 5       do conv until ce;
 6 conv: mr=mr+mx0*my0(SS), mx0=dm(i1,m1), my0=pm(i5,m5);
 7       mr=mr+mx0*my0(SS);
 8       sr = lshift mr0 by -1 (lo);
 9       sr = sr or ashift mr1 by -1 (hi);
10       i1=dm(convol_ptr_);
11       dm(i1,m1)=mr0;
```

In regel 1 wordt het indexregister *i1* gelijk gemaakt aan de pointervariabele *stripe_ptr*. De streeppunten worden per *area* opgeslagen in een array, genaamd *stripe[area]*. De pointervariabele stripe_ptr wijst in het array stripe[area] naar het i^e element.
De filtercoëfficiënten van het DOG-filter zijn opgeslagen in het array *dog_kernel*.
In regel 2 wordt het indexregister i5 gelijk gemaakt aan het adres van het eerste element van het *dog_kernel* array, namelijk het element *dog_kernel[0]*.
Het array stripe[area] is gelegen in het data geheugen van de ADSP-2100 (data memory). Echter het array dog_kernel is gelegen in het programma geheugen (program memory, data section).

Het resultaat van de convolutie komt in eerste instantie terecht in het register *mr*. Dit register is het zogenaamde *result register* van de *multiplier/accumulator block (MAC)*. Dit register wordt geïnitialiseerd op de waarde 0 in regel 3.
Tevens wordt in de programmaregel de inhoud van de geheugenplaatsen,

waar de indexregisters i1 en i5 naar wijzen, overgebracht naar de MAC-registers *mx0* en *my0*. Dit gebeurt m.b.v. de zogenaamde *move instructies dm* en *pm*. De instructie *dm* dient om een geheugenplaats in het data geheugen uit te lezen. De instructie *pm* om een geheugenplaats in het programma geheugen uit te lezen. Tevens worden de indexregisters i1 en i5 opgehoogd met de inhoud van de *modify registers m1* en *m5*. De inhoud is bij beide modify registers gelijk aan 1.

In regel 4 krijgt het counter register (*cntr*) de waarde 15. De inhoud van dit register bepaalt het aantal malen dat de programmalus - dit zijn de instructies in regel 6 - wordt doorlopen.
De vermenigvuldiging van de twee laatste elementen uit het stripe_array en het dog_kernel array wordt uitgevoerd in regel 7 van het bovenstaande programma.

Het MAC-block is eigenlijk ontworpen om getallen te vermenigvuldigen die in de *fixed point format* zijn genoteerd. Wanneer twee getallen, genoteerd in de *two's complement format*, worden vermenigvuldigd, dan is het resultaat een factor 2 te groot. Men verkrijgt het juiste resultaat door de inhoud van het *mr* register over één bit naar rechts te schuiven. Deze operatie wordt uitgevoerd in het *shifter block* (zie regel 8 en 9)

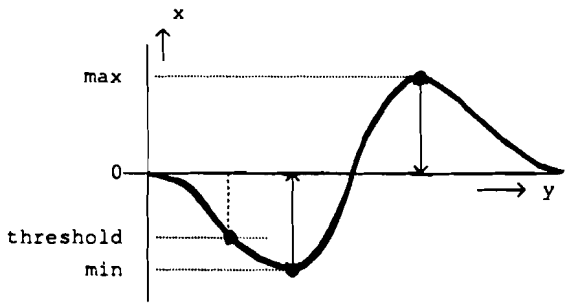
In regel 10 en 11 staan de instructies die er voor zorgen dat de inhoud van het register *mr* wordt overgebracht naar het adres waar de pointer *convol_ptr* naar wijst.

Voor meer informatie over de assembly-instructies verwijs ik u naar de ADSP-2100 User's Manual [1].

5.6. De detectiemodule

Een plotselinge of meer geleidelijke overgang in de stripe dataset resulteert in een opslingering in de DOG-filterresponsie. Deze opslingering vertoont grote gelijkenis met de stapresponsie van het DOG-filter.
Telkens wanneer een dergelijke opslingering optreedt in de filterresponsie, dan is het de taak van de detectiemodule om te beslissen of deze correspondeert met een naad die valt binnen de categorie gezochte naden.

Wanneer in de stripe dataset een sprongetje optreedt in positieve (x-) richting dan resulteert dat in een responsie zoals getoond in figuur 5.29.



sum = |max - min|
diff = |max + min|

stapresponsie voldoet wanneer geldt:

diff < diff_max
sum_min < sum < sum_max

fig. 5.29: Opslingering in convol dataset.

De responsie is eerst negatief en daarna positief. Indien de responsie kleiner wordt dan een bepaalde drempel (*threshold*) dan is het interessant om de beweging van de responsie verder te analyseren. Eerst worden het minimum en het maximum van de responsie bepaald en daaruit worden de grootheden *sum* en *diff* (difference) berekend. Deze zijn als volgt gedefiniëerd:

$$sum = |max - min|$$

$$diff = |max + min|$$

De grootheid *sum* is het verschil tussen de minimale en maximale responsie. De grootheid *diff* is een maat voor de symetrie van responsie t.o.v. het nulpunt tussen het minimum en maximum.
Volgens de detectiemodule hebben we te maken met een naadpunt wanneer in de convol datastructuur een slingerbeweging optreedt waarvoor geldt:

$$sum\_min < sum < sum\_max$$

$$diff < diff\_max$$

Dit naadpunt wordt gedefiniëerd als het punt dat halverwege de overgang in de stripe datastructuur ligt. Dit naadpunt ligt altijd in een gebied van 8 pixels breed vóór de plaats waar het minimum ligt. De y-coördinaat van het minimum is gelijk aan *y_min* (fig 5.30).
De x-coördinaat van het naadpunt wordt gedefiniëerd als zijnde gelijk aan:

$$x\_half = \frac{\texttt{stripe[area][y\_min]} + \texttt{stripe[area][y\_min - 8]}}{2}$$

Indien we te maken hebben met een stripe van het type *a* (§ 5.3) dan wordt in de stripe vanaf de y-coördinaat y_min het eerste streeppunt gezocht waarvan x-coördinaat groter is dan x_half. Bij stripes van het type *b* wordt het eerste streeppunt gezocht waarvan de x-coördinaat kleiner is dan x_half.

6. Tests

6.1. Inleiding

In hoofdstuk 5 is de patroonherkenningsalgoritme beschreven die wordt gebruikt om de positie van de naad in het camerabeeld te bepalen. Deze algoritme dient te worden geïmplementeerd op de ADSP-2100 digitale signaal processor.
Voor de ontwikkeling van deze algoritme werd gebruik gemaakt van het *general purpose* beeldverwerkingssysteem van Imaging Technology, Series 150/151. Dit systeem biedt een aantal grafische faciliteiten die het mogelijk maken om het effect van bepaalde beeldbewerkingstechnieken zichtbaar te maken. Voor meer informatie over dit systeem verwijs ik u naar appendix B. De algoritme is geschreven in de programmeertaal C en daarna gecompileerd m.b.v. de Microsoft C-compiler.

Het programma, dat op het beeldverwerkingssysteem van Imaging Technology is ontwikkeld, dient te worden omgezet in machine-instructies voor de ADSP-2100. Instructies, die dienen om het beeldverwerkingssysteem te besturen, worden uit het programma verwijderd. Andere instructies, zoals instructies voor de communicatie met de coördinatengenerator kaart, dienen te worden toegevoegd. Daarna kan het C-programma vertaald worden m.b.v. van de C-compiler, die speciaal door Analog Devices voor de ADSP-2100 is ontwikkeld. De code, die deze compiler genereert, kan worden uitgetest op de ADSP-2100 signaalprocessor op het evaluation board.

6.2. Beeldverwerkingssysteem: Image Processor Series 150/151

Figuur 6.1 toont de testopstelling met het beeldverwerkingssysteem van Imaging Technology, Series 150/150. Het videosignaal, afkomstig van de camera, wordt voorbewerkt m.b.v. een pulsbreedte filter, voordat het aan het beeldverwerkingssysteem wordt aangeboden. De beeldverwerkingsalgoritme wordt uigevoerd op de AT host computer. De resultaten van deze algoritme kunnen zichtbaar worden gemaakt op een monitor.

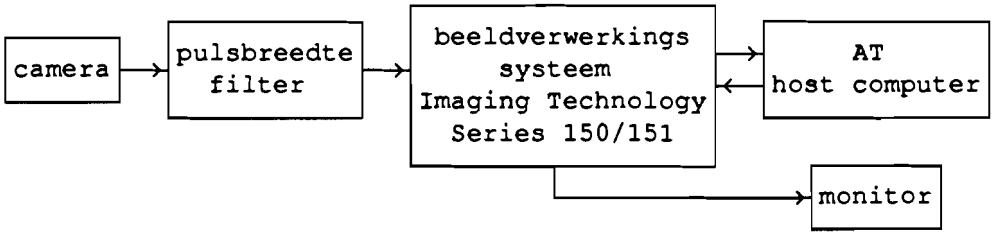


fig. 6.1: Testopstelling.

Volvo Car Nederland B.V. stelde een autoportier beschikbaar om de algoritmen te testen. De naden in dit portier zijn, qua vorm, representatief voor de naden aan de onderzijde van de carrosserie. De naden, die moeten worden afgedekt, zitten voornamelijk aan de rand van het portier. In figuur 6.2 zien we de projector, die op de naden een aantal lichtrepen projecteert. In die figuur is tevens is de camera te zien die het streepjespatroon opneemt.

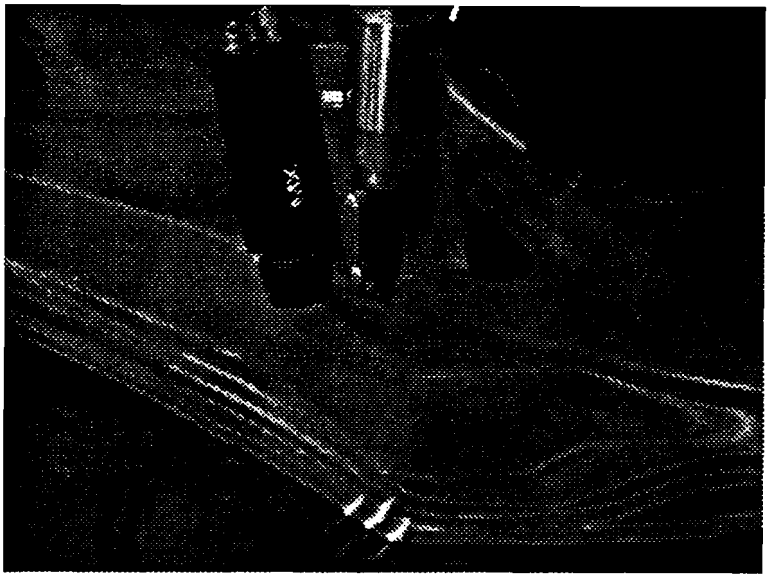


fig. 6.2: Rand van het autoportier belicht m.b.v. gestructureerd licht.

Figuur 6.3 toont een dwarsdoorsnede van het portier. De gezochte naden bevinden zich op de plaatsen waar het metaal is omgevouwen. Daar is immers de kans op roestvorming het grootst.

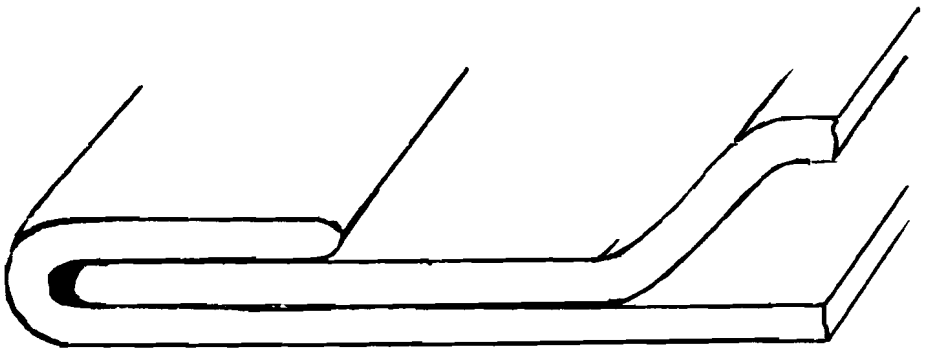


fig. 6.3: Dwarsdoorsnede van de rand van het portier.

Figuur 6.4 toont het camerabeeld met daarin het streepjespatroon dat op het werkstuk wordt geprojecteerd. Ook is in die figuur de responsie van het pulsbreedte filter weergegeven. Deze responsie heeft een vertraging van bijna 4 μsec t.o.v. de lichtstrepen.

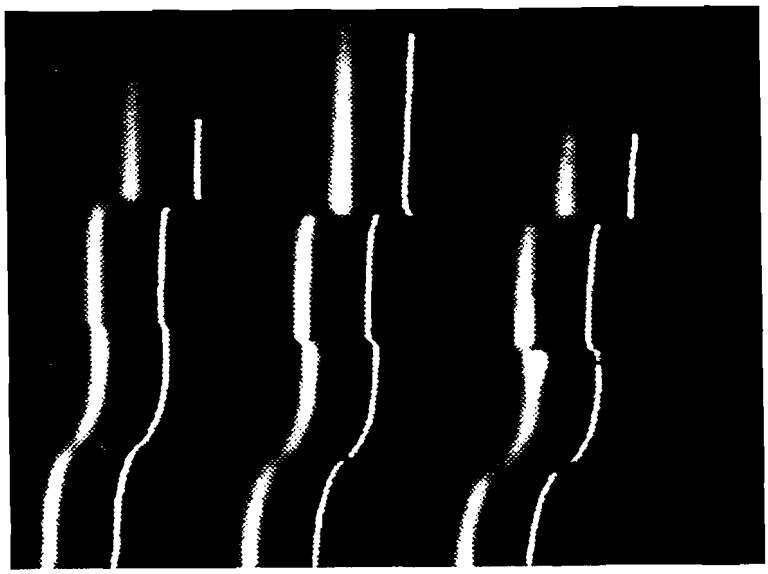


fig. 6.4: Camerabeeld en dogint pulsen.

Figuur 6.5 toont de beeldinformatie ná distributie. De verticale strepen in het beeld geven de grenzen aan van de gebieden waarbinnen de lichtstreep wordt verwacht (areas). In elk gebied wordt de lichtstreep beschreven d.m.v. een stripe dataset. Merk op dat de ongedefiniëerde stukken in de verzameling streeppunten zijn opgevuld (oftewel: gereconstrueerd).

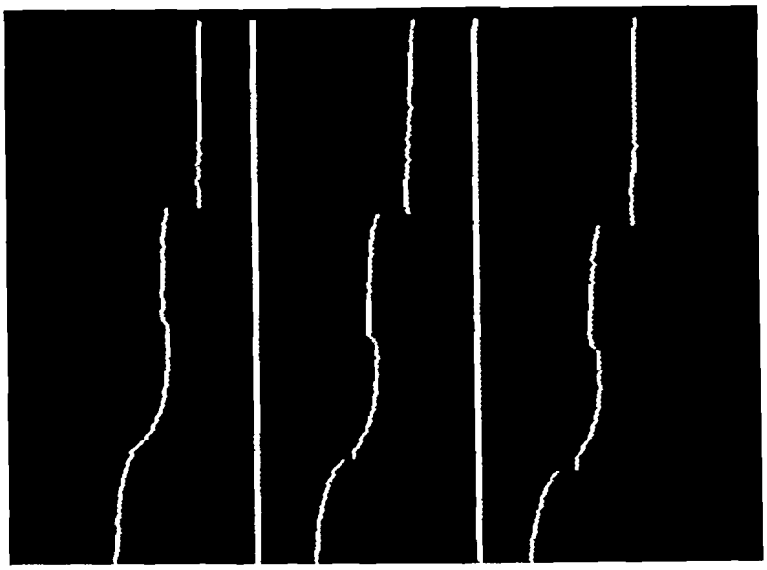


fig. 6.5: Beeldinformatie ná distributie.

De stripes worden gefilterd m.b.v. het discrete DOG-filter. Figuur 6.6 toont per stripe de DOG-filter-responsie.

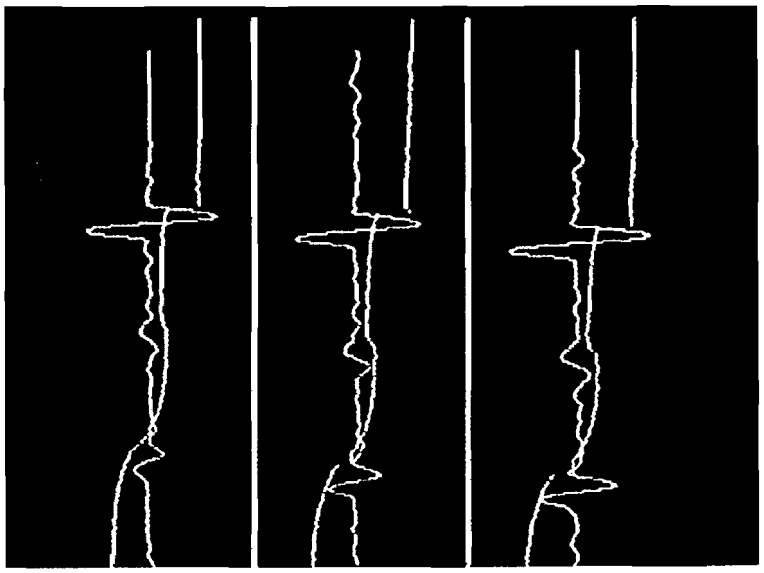


fig 6.6: DOG-filter responsie.

Op de plaats waar in het stripe-array een sprong optreedt, ontstaat in de responsie van het DOG-filter een opslingering. De taak van de detectiemodule is om na te gaan of deze opslingering correspondeert met de gezochte naad. Indien het een naad betreft, die behoort tot de categorie gezochte naden, dan worden van het naadpunt de x-, y-, en z-coördinaat bepaald. De gevonden naadpunten zijn in figuur 6.7 d.m.v. een pijltje gemarkeerd.

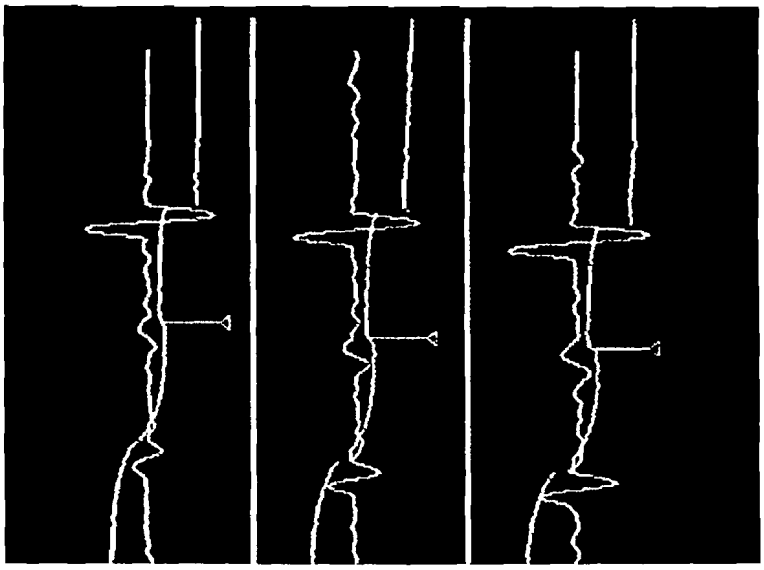


fig. 6.7: De gemarkeerde punten in de stripes zijn de plaatsen waar de naad optreedt.

Prestaties

Onder de camera in de figuren 6.4 tot en met 6.7 ligt een autoportier. De breedte van de scène is gelijk aan 5 cm. Het plaatstaal van het autoportier heeft een dikte van ± 0,5 mm. Het werkstuk wordt door de projector belicht onder een hoek van 30° t.o.v. de as van de camera. Het beeldverwerkingssysteem verdeelt een beeldlijn in het analoge videosignaal in 512 pixels.
Volgens bovenstaande gegevens komt de sprong in de lichtstreep, op de plaats waar de naad optreedt, overeen met ongeveer 3 pixels.

In het geval dat er in de scène 3 naadpunten voorkomen (één naadpunt per lichtstreep), wordt in ongeveer 70% van de gevallen alle drie de naadpunten gevonden. In ongeveer 20% van de gevallen worden er twee naadpunten gevonden en in 10% wordt één van de drie naadpunten gevonden. Gegeven dat we hier te maken hebben met een zeer kleine sprong in de lichtstreep (*slechts 3 pixels*) is de betrouwbaarheid van het beeldverwerkingssysteem redelijk te noemen.
Door de *procesparameters* van de beeldverwerkingssoftware (sum_min, sum_max, etc.) anders te kiezen kunnen waarschijnlijk nog betere resultaten worden bereikt. Naar de juiste instelling van de procesparameters zal nog extra onderzoek worden verricht.

Wanneer onder de camera een werkstuk wordt gelegd met daarin een overlapnaad, waarbij het plaatstaal een dikte heeft van ± 1,0 mm,

dan worden in bijna 100% van de gevallen de drie naadpunten gevonden. (Merk op dat het plaatstaal aan de onderkant van de carrosserie een dikte heeft van ongeveer 1 mm)

De tijd, die het beeldverwerkingssysteem nodig heeft om één beeld te verwerken, komt overeen met 650 msec.

De belichting is bij de tests steeds zodanig ingesteld dat in de lichtstreep zo weinig mogelijk onderbrekingen optreden. Deze onderbrekingen zijn er meestal de oorzaak van dat er foute naadpunten worden gevonden.
Daar onderbrekingen in de lichtstrepen niet altijd kunnen worden vóórkomen, zal de reconstructie van de lichtstreep om een meer geavanceerdere manier moeten worden uitgevoerd. Dit zal echter ten koste gaan van de snelheid van het beeldverwerkingssysteem.

De mate waarin onderbrekingen voorkomen in de lichtstreep is sterk afhankelijk van:

- de kwaliteit van de belichting en de prestaties van het pulsbreedte filter (DOG-filter)
- de vorm van het werkstuk

6.3. Tests: ADSP-2100

De betrouwbaarheid, waarmee de patroonherkenningsalgoritme de naadpunten in het camerabeeld zoekt, is getest op de Image Processor Series 150/151 van de firma Imaging Technology.
Naast de betrouwbaarheid zijn we ook geïnteresseerd in de snelheid waarmee de digitale signaalprocessor ADSP-2100 de algoritme uitvoert.

Time-sharing

Gedurende de 20 msec dat de streeppunten van één halfbeeld ingelezen worden, is het hoofdprogramma bezig om de streeppunten te verwerken, die tot dan toe zijn ontvangen. Dit *hoofdproces* wordt onderbroken op het moment dat een nieuw streeppunt door de interrupt serviceroutine moet worden ingelezen.

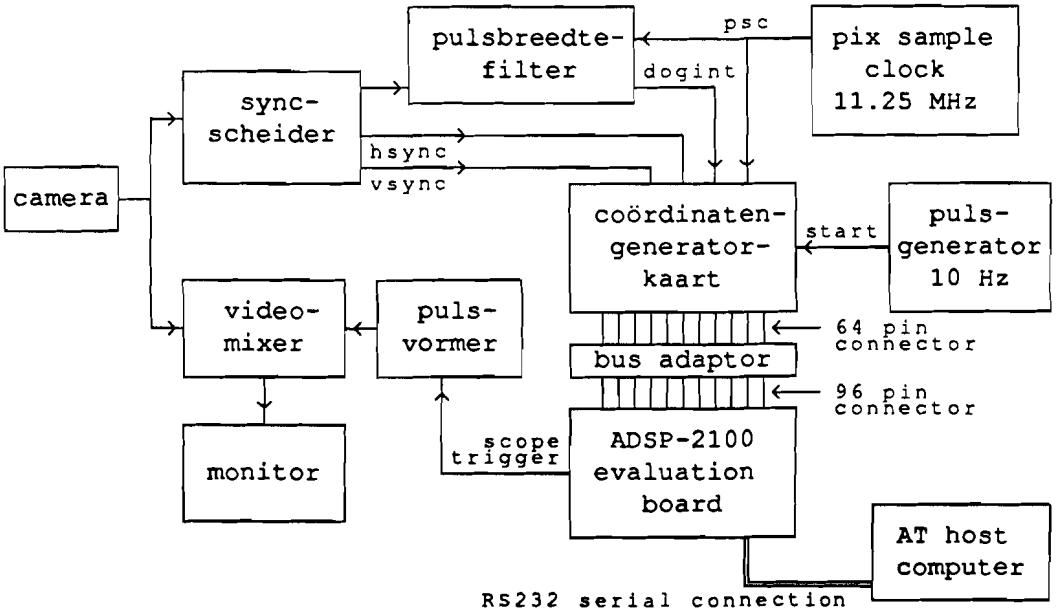


fig 6.8: Proefopstelling 1.

In begin en op het eind van de serviceroutine wordt een instructie uitgevoerd, die een pulsje genereert op de *scope trigger* uitgang van het evaluation board. Dit is de instructie:

```
dm(SCOPE_TRIGGER) = si;
```

Het pulsje, dat op de scope trigger uitgang verschijnt, wordt verlengd m.b.v. een pulsvormer en daarna gemixed in het videobeeld dat de camera afkomt (zie proefopstelling 1, figuur 6.8).
Figuur 6.9 toont het beeld op de monitor.

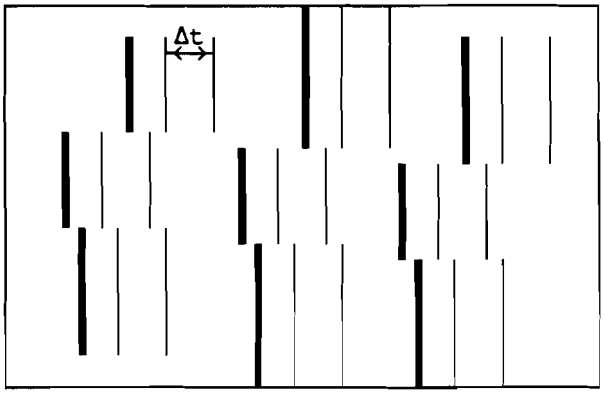


fig. 6.9: Monitor: ▌ : Lichtstreep in het camerabeeld,
▏ : Scope trigger output,
Δt = 2.375 μsec.

In de interrupt service routine worden 19 instructies uitgevoerd. De ADSP-2100 op het evaluation board voert deze instructies uit in een tijd van 2.375 μsec (één instructie komt overeen met 125 nsec). Wanneer het pulsbreedte filter per beeldlijn (64 μsec) gemiddeld 3 interrupts genereert dan is de verhouding tussen de tijd, die nodig is voor de uitvoer van de interrupt service routine, en de tijd die het hoofdproces krijgt toegewezen per beeldlijn gelijk aan:

$$3 \times 2.375 : (64 - 3 \times 2.375) \approx$$
$$7.1 : 56.8 \approx$$
$$1 : 8$$

Snelheid beeldverwerkingssysteem

In figuur 6.10 is de proefopstelling weergegeven waarmee de snelheid van de patroonherkenningsmodule wordt gemeten.

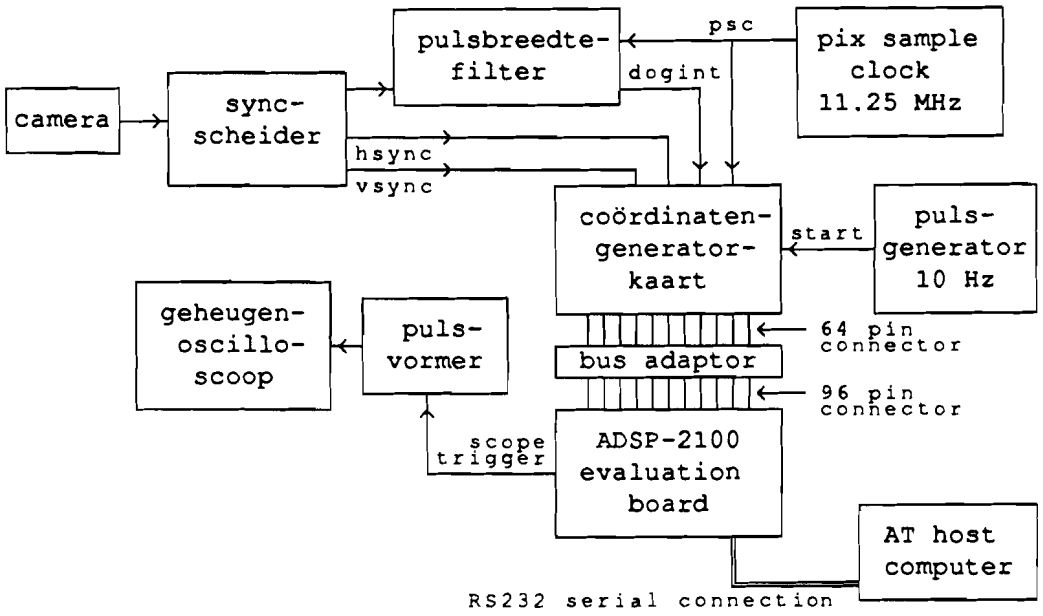


fig 6.10: Proefopstelling 2.

De functie send_scope_trigger_pulse() zendt een puls van 0,5 µsec naar de trigger scope uitgang van het evaluation board en wacht daarna gedurende een periode van 0,5 µsec.
Deze functie wordt aangeroepen:

- nadat de eerste vsync puls is gedetecteerd (fase 3)
- nadat de tweede vsync puls is gedetecteerd (fase 4)
- voordat de functie add_tails() wordt uitgevoerd (fase 5)
- voordat de functie filter_module() wordt uitgevoerd (fase 6)
- voordat de functie detection_module wordt uitgevoerd (fase 7)
- nadat de laatsgenoemde functie is uitgevoerd (fase 8)

Figuur 6.11 toont het beeld van een geheugen-oscilloscoop. In dit beeld zijn de pulsen zichtbaar gemaakt die verschijnen op de trigger scope uitgang van het evaluation board.

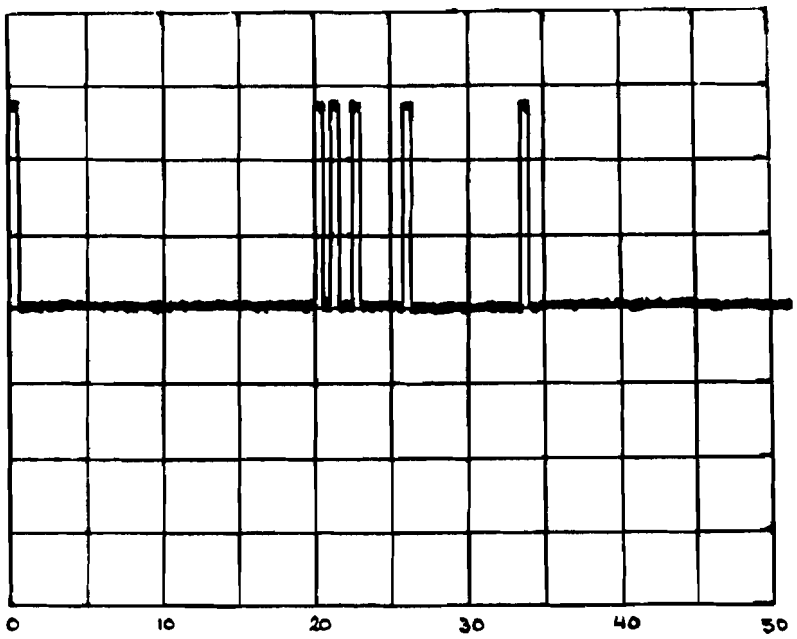


fig. 6.11: Pulsen op de uitgang van de scope trigger output zichtbaar gemaakt op een geheugen-oscilloscoop. Timebase: 5 msec/division

Volgens figuur 6.11 heeft de patroonherkenningsmodule één halfbeeld binnen 40 msec verwerkt.
In tabel 6.1 zijn per module de verwerkingstijd gegeven.

Tabel 6.1: Verwerkingstijd per module

module	fase	verwerkingstijd (msec)
distribution_module() onderbroken door de service-routine van interrupt int0	3 + 4	20
add_tails()	5	0,5
filter_module()	6	5
detection_module()	7	8

7. Conclusies en aanbevelingen

7.1. Conclusies

De belangrijkste conclusie van het project *"ontwerp van een patroon-herkenningsmodule voor een snel naadvolgsysteem"* is dat het mogelijk is om een beeldverwerkingssysteem te realiseren dat binnen 50 msec de positie van een overlapnaad in een werkstuk bepaald. (Het materiaal waaruit de carrosserie is opgebouwd is plaatstaal met een dikte van 0.8 mm).
Dit beeldverwerkingsysteem heeft de volgende kenmerken:

- Het werkstuk wordt belicht met een streeppatroon, bestaande uit drie lichtstreepjes, dat direct de geometrische vorm van naad zichtbaar maakt (gestructureerd licht).
- Het videosignaal wordt *realtime* voorbewerkt m.b.v. een analoog pulsbreedte filter (DOG-filter).
- Om de naadpunten te vinden wordt op de lichtstrepen in de scène een edge enhancement operatie uitgevoerd m.b.v. een discreet DOG-filter.

De belichting van het sensorsysteem moet zodanig zijn ingesteld dat in de lichtstreep zo weinig mogelijk onderbrekingen optreden. Indien dit niet het geval is, dan is de kans groot dat de reconstructie van de niet-gedefiniëerde stukken in de lichtstrepen niet goed wordt uitgevoerd. Het gevolg is dat valse naadpunten worden gevonden.

Het succes van het beeldverwerkingssysteem is te danken aan de selectiviteit van het DOG-filter. Dit filter wordt enerzijds gebruikt als pulsbreedte filter om het sensorsignaal (videosignaal) voor te bewerken. Anderzijds wordt het gebruikt als edge enhancement filter om de naadpunten in de lichtstrepen te accentueren.

In de literatuur zijn twee andere beeldverwerkingssystemen beschreven. Bij beide wordt tevens gebruik gemaakt van gestruktureerd licht. Het systeem, beschreven door Clocksin [6 en 7], is voor onze toepassing te traag. Het andere systeem, beschreven door Niepold [10], is snel genoeg maar is niet betrouwbaar.

Het beeldverwerkingssysteem, dat bij dit project werd gerealiseerd, is uitgevoerd op een digitale signaalprocessor. Er kan tijdswinst worden gemaakt door tijdens het inlezen van de videoinformatie van één half-beeld reeds te beginnen met de verwerking van de tot dan toe ingelezen informatie.

Het inlezen van de videoinformatie wordt bestuurd m.b.v. zogenaamde

synchronisatiesignalen op de interruptingangen van de digitale signaalprocessor. Bij dit proces treden fouten als gevolg van overspraak op de genoemde interruptingangen. Om deze fouten te vermijden dienen de ingangen te worden geprogrameerd in de zogenaamde level sensitive mode.

7.2. Aanbevelingen

Parallelle processing

Op dit moment is de complete patroonherkenningsmodule geïmplementeerd op één digitale signaalprocessor. Zoals figuur 7.1 toont kan een aantal processen in de patroonherkenningsmodule worden gesplitst en uitgevoerd op parallel werkende digitale signaalprocessoren.

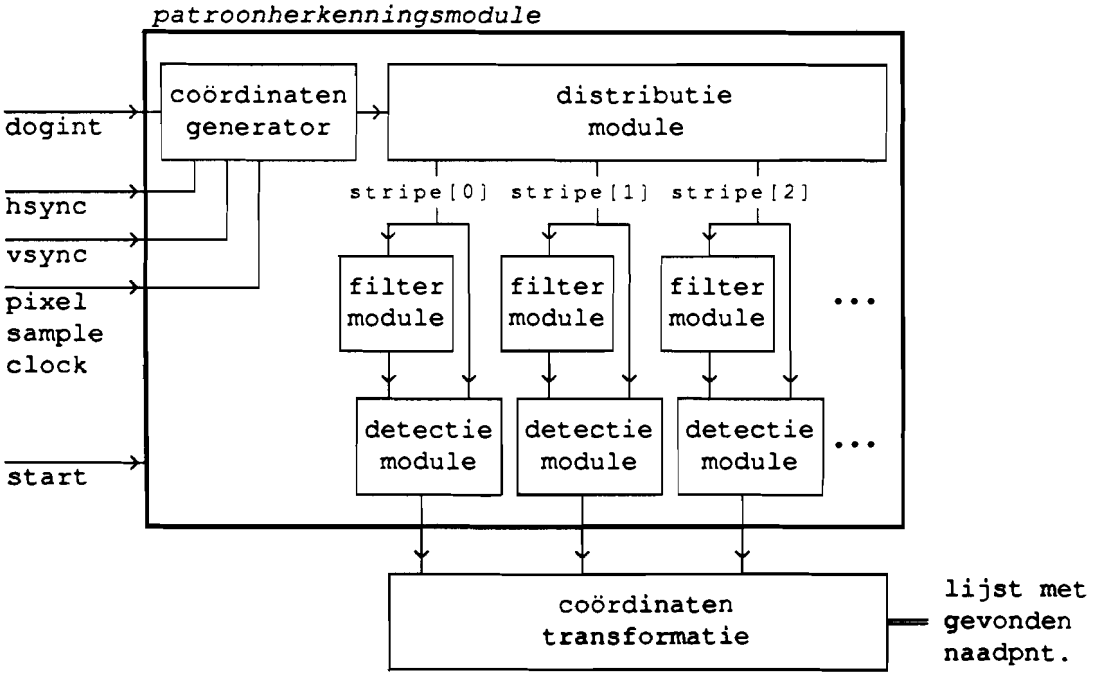


fig. 7.1: Patroonherkenningsmodule met paralelle processen.

Het voorstel is een configuratie met daarin één *master* processor met een aantal *slave* processoren. De master processor zorgt voor de dis-

tributie en zet per area een slave processor aan het werk om de filter operatie, de detectie en de coördinatentransformatie uit voeren. Het voordeel van deze configuratie is dat de snelheid waarmee de patroonherkenningsmodule de stripes verwerkt onafhankelijk is van het aantal lichtstrepen in het camerabeeld.

Voor de slave processoren gaat de voorkeur uit naar de ADSP-2101 [4] in plaats van de ADSP-2100. De ADSP-2101 digitale signaalprocessor is *upwards compatible* met de ADSP-2100. D.w.z. dat de programmatuur, die is ontwikkeld voor de ADSP-2100, kan worden overgedragen naar de ADSP-2101. De omkering geldt echter niet.
Een voordeel van de ADSP-2101 t.o.v. de ADSP-2100 is dat de ADSP-2101 een intern programma geheugen en een intern data geheugen heeft (beide RAM). De grootte van het programma geheugen is 2K, de grootte van het data geheugen is 1K. Het data geheugen is voldoende groot voor één stripe- en één convol-array.
Een ander voordeel is de gunstige prijs. De ADSP-2101 is ongeveer 30% goedkoper dan de ADSP-2100, dit ondanks het feit dat er op chip geheugen aanwezig is. De prijs bij dit soort chips wordt veelal bepaald door het aantal pennen op de chip. Deze is bij de ADSP-2101 kleiner dan bij de ADSP-2100.

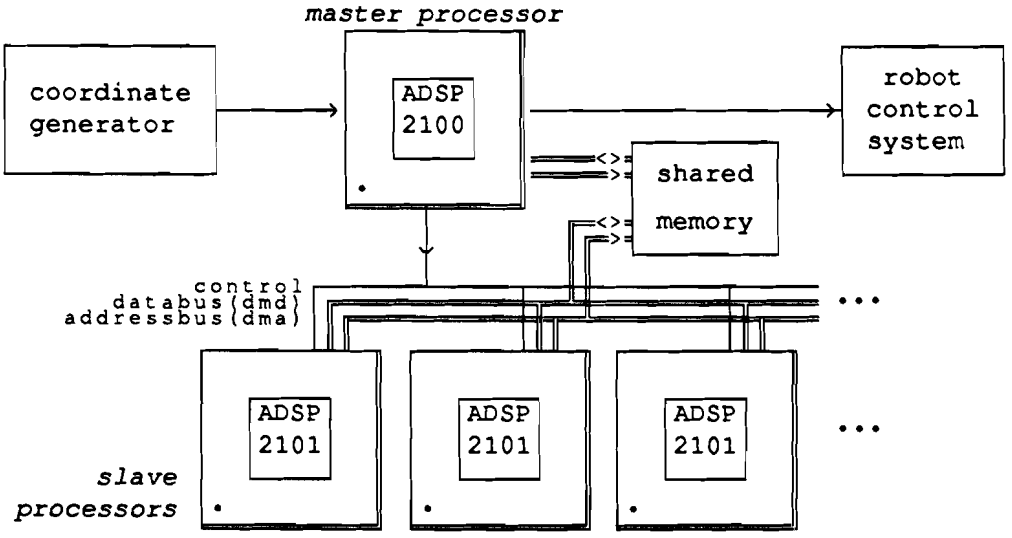


fig. 7.2: Configuratie met meer signaal processoren.

Extra areas

Rondom de plaats waar het beeldverwerkingssysteem een lichtstreep verwacht wordt een area gedefiniëerd. Een probleem treedt op wanneer een lichtstreep schuin in het camerabeeld ligt en de overgang net op grens

tussen twee areas komt te liggen. In dat geval zal de stripe niet worden gedetecteerd.
Dit probleem kan worden opgelost door rondom de grenzen tussen de areas nieuwe areas te definiëren zoals weergegeven in figuur 7.3.

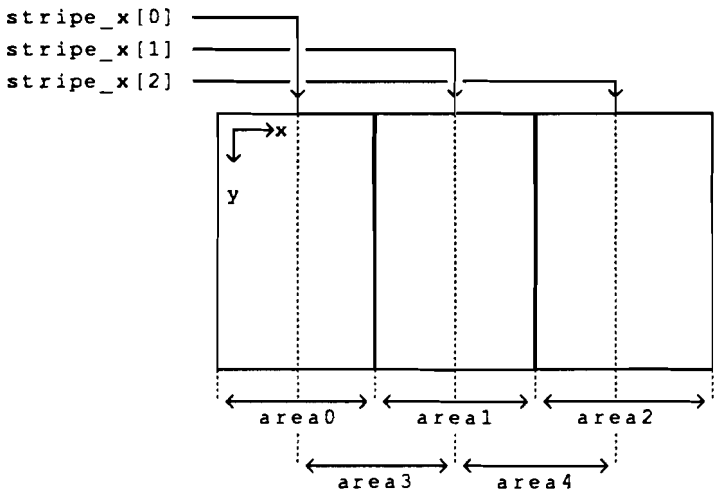


fig. 7.3: Extra areas rondom de grenzen tussen area0 en area1 en tussen area1 en area2.

Wanneer bijvoorbeeld een naadpunt op de grens tussen area0 en area1 ligt dan zal er geen overgang optreden in de stripe datasets van de genoemde areas. Echter de overgang treedt wel op in de stripe dataset van area3.

Selector

De patroonherkenningsmodule genereert een lijst met daarin de coördinaten van de gevonden naadpunten. Soms zullen in deze lijst één of meer punten voorkomen die niet tot de naad behoren. De oorzaak hiervan zijn bijvoorbeeld reflecties op het werkstuk waardoor de patroonherkenningsmodule in verwarring wordt gebracht.
Deze 'valse' punten dienen uit de lijst te worden verwijderd. Hierbij kan gebruik worden gemaakt van voorkennis over de ligging van de gezochte naad. De zogenaamde selector voert deze taak uit (figuur 7.4). Op grond van de voorkennis die het systeem heeft over de gezochte naad, selecteert deze module de juiste naadpunten uit de verzameling gevonden *kandidaat*-naadpunten.

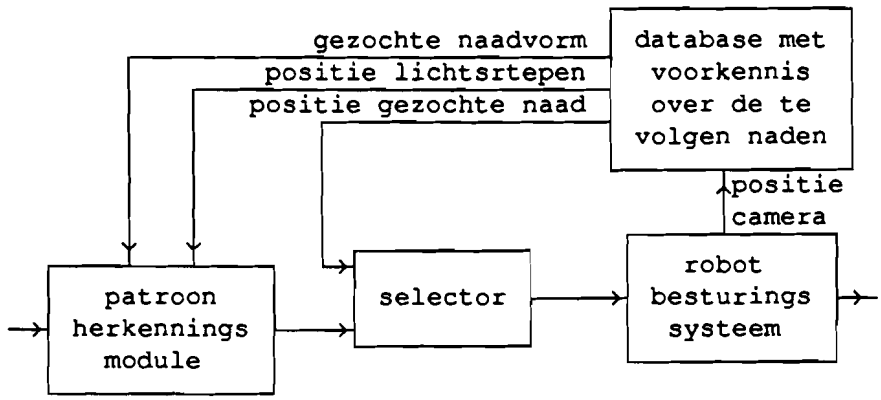


fig. 7.4: De selector selecteert de juiste naadpunten uit de lijst gevonden naadpunten. De selectie vindt op grond van voorkennis over de *verwachte* ligging van de naad.

Hoe meer lichtstrepen worden gebruikt in het streepjespatroon waarmee het werkstuk wordt belicht, des te meer naadpunten zullen worden gevonden en des te beter zal de ligging van de naad zijn beschreven.

Schellekens [12] beschrijft een systeem dat in staat is om V-vormige naden op te speuren in een werkstuk m.b.v. gestructureerd licht. Ook Schellekens maakt gebruik van een groot aantal lichtstrepen om de valse naadpunten te onderscheiden van echte naadpunten.

FIFO buffer

Nadat een dogint pulsje is ontvangen voert de digitale signaalprocessor de interrupt service routine uit die de coördinaten van de streeppunten uit het XREG en het YREG register inleest en deze opslaat in de pixel-tabel. Het aantal instructies in de service routine is gelijk aan 19 (§ 6.3). Terwijl deze routine wordt uitgevoerd is de processor ongevoelig voor nieuwe dogint pulsjes. Wanneer de tijdsduur tussen twee dogint pulsjes te klein is, dan zal het tweede pulsje niet worden gedetecteerd.
De oplossing is om de coördinaten niet in de ADSP-2100 in te lezen via het interrupt mechanisme, maar de coördinaten op te slaan in een *FIFO* buffer (*first in, first out*). Het FIFO buffer maakt tevens de constructie met de *pixel_table* datastructuur overbodig.

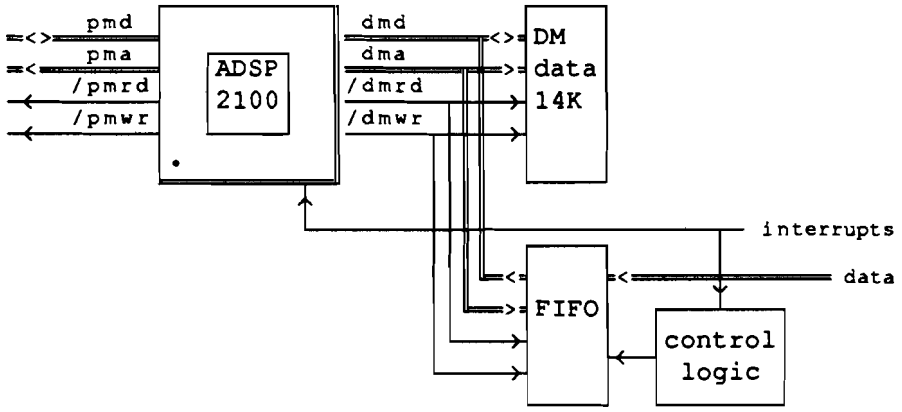


fig. 7.5: Constructie met FIFO buffer.

Software downloading

De machinecode voor de ADSP-2100 wordt op dit moment via een seriële lijn naar het programmageheugen van deze processor overgebracht.
De communicatie tussen het evaluation board en de host-computer verloopt uiterst traag. Het duurt ongeveer 15 minuten voordat de lijst machine-instructies is overgezonden.

Een andere student is op dit moment bezig met de realisatie van een interface voor het overbrengen van informatie van de ADSP-2100 naar de hostcomputer. Deze interface dient te worden uitgebreid zodanig dat ook informatie van de hostcomputer kan worden overgebracht naar de signaalprocessor.
Telkens wanneer de ADSP-2100 wordt opgestart dient deze een lijst machine instructies via de interface over te brengen naar zijn programma geheugen. De communicatiesoftware hiervoor wordt opgeslagen in een stukje ROM in het programma geheugen. De eerste instructie van deze software dient te liggen op adres 4. Na een *reset* cyclus is de instructie op dit adres de eerste die door de signaalprocessor wordt uitgevoerd.

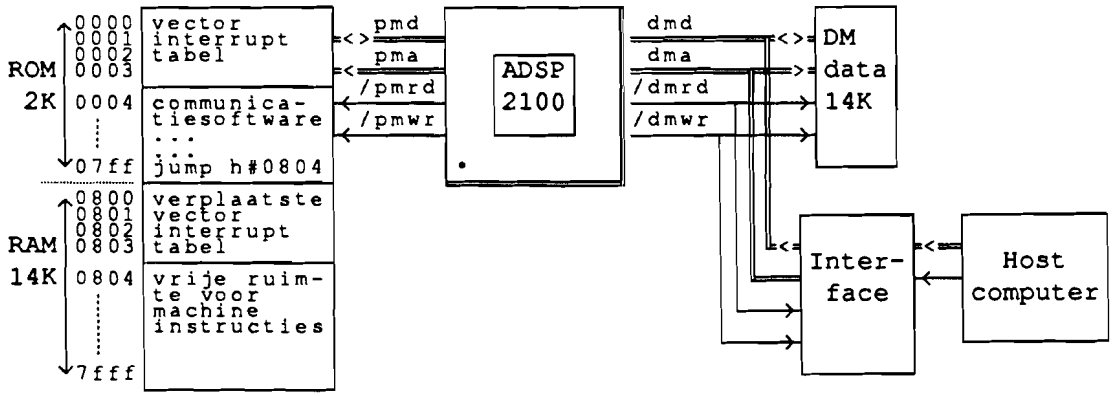


fig. 7.6: Software downloading vanuit de host computer via het interface naar het programmageheugen in de adres-ruimte: h#0804 - h#7fff.

Om praktische redenen zal de vector interrupt tabel ook in het stukje ROM-geheugen gelegen zijn. Echter deze tabel kan worden verplaatst door in die tabel jump instructies te plaatsen naar een andere tabel. Het gevolg is dat bij iedere interrupt een extra jump instructie moet worden uitgevoerd, echter dit zal niet tot noemenswaardige problemen leiden.

Backplane

Eventuele problemen met het backplane kunnen worden vermeden door de hardware, bestaande uit coördinatengenerator en interface naar de host computer, uit te voeren op één print.

Alfabetische indexlijst

Literatuuropgave

[1] Analog Devices, *ADSP-2100 User's manual, architecture;* Analog Devices Inc., Norwood, Massachussets, USA/ 1988;

[2] Analog Devices, *ADSP-2100 Cross-software manual, programming reference;* Analog Devices Inc., Norwood, Massachussets, USA, 1988;

[3] Analog Devices, *ADSP-2100 Evaluation board manual;* Analog Devices Inc., Norwood, Massachussets, USA, 1988;

[4] Analog Devices, *ADSP-2101 and ADSP-2102 User's manual, (preliminary);* Analog Devices Inc., Norwood, Massachussets, USA, 1988;

[5] Bogers, C.J.M., *Ontwikkeling en Implementatie in hardware van een diffenrence of Gaussian filter voor een naadzoeksysteem;* Master Thesis, Department of Electrical Engineering, Eindhoven University of Technology, August 1989;

[6] Clocksin W.F., P.G. Davey, C.G. Morgan, A.R. Vidler, *Progress in visual feedback for robot arc-welding of thin sheet steel;* Proceedings 2nd International Conference on Robot Vision and Sensory Controls, November 1982;

[7] Clocksin W.F., J.S.E. Bromley, P.G. Davey, C.G. Morgan, A.R. Vidler, *Implementation of model-based visual feedback for robot arc welding of thin sheet steel;* The International Journal of Robotics, Vol. 4, No. 1, Spring 1985;

[8] Fu K.S., *Syntactic pattern recognition, applications/ Springer Verlag;* Berlin 1977, ISBN 3-540-07841-x, Chapter 2: *Peak recognition in waveforms* (author: S.L. Horowitz);

[9] Kernigham B.W., Ritchie D.M., *The C programming language;* Englewood Cliffs, N.J., Prentice-Hall, 1978;

[10] Niepold R., G. Struck, *Co-operation of a structured light sensor with an industrial robot for arc welding of thin sheet;* Proceedings SPIE Internatinal Society on Optical Engineering, Vol. 595, pp 284-291, 1985;

[11] Shear D., *EDN's DSP Benchmarks;* EDN, pp 126-144, September 19, 1988;

[12] Schellekens R.A.A.J., *Naadzoeken met behulp van VISION technieken;* Master Thesis, Department of Electrical Engineering, Eindhoven University of Technology, June 1988;

[13] Strand T.C., *Optical three-dimensional sensing for machine vision;* Optical Engineering, Vol. 24, No. 1, 1985;

[14] Vliet R. van, *Dictaat beeldverwerking;* group measurement and control; Department of Electrical Engineering, Eindhoven University of Technology, 1988;

Appendices

Appendix A: De CCD-camera van HTH
B: General purpose beeldverwerkingssysteem:
Image Processor Series 150/151
C: Memory map ADSP-2100
D: Coordinate generator
E: Coordinate generator, electric design
F: Prototyping connector ADSP-2100
G: Bus backplane connector
H: Application backplane connector
I: ADSP-2100 Cross-software
J: Listing image processing software:
general purpose Image Processor Series 150/151
K: Listing image processing software:
ADSP-2100 system
L: Pattern recognition software diskette

Appendix A: De CCD-camera van HTH

Het werkstuk wordt belicht d.m.v. gestructureerd licht. Een streepjespatroon, bestaande uit een aantal evenwijdige lichtstrepen, wordt op het metaaloppervlak geprojecteerd, zodanig dat de lichtrepen loodrecht op de richting van de naad staan. Het streepjespatroon wordt opgenomen m.b.v. een standaard video-camera, in dit geval een CCD-camera van de firma HTH.
Het lichtgevoelige deel in dit soort camera's is een zogenaamde *CCD-chip* (charge coupled device). Op deze chip bevinden zich een groot aantal lichtgevoelige elementen die in een matrix zijn gerangschikt (604 x 576 elementen). Door het opvallende licht worden in deze elementjes ladingsdragers vrijgemaakt. Een lenzenstelsel projecteert een lichtbeeld op het lichtgevoelige deel van de CCD-chip. Dit lichtbeeld wordt omgezet in een corresponderend ladingspatroon in de matrix.

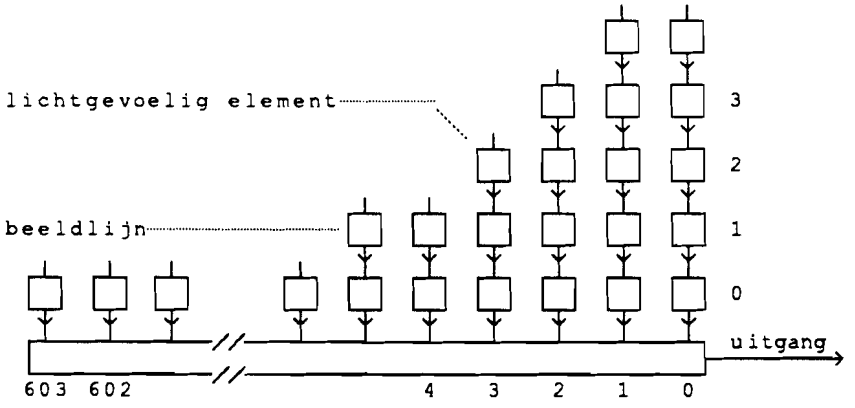


fig. a.1: Matrix van lichtgevoelige elementen op de CCD-chip.

Een reeks lichtgevoelige elementen, die horizontaal op één rij liggen, wordt een beeldlijn genoemd (zie figuur a.1). Afwisselend wordt de lading, die in de even en de oneven beeldlijnen is vrijgemaakt, naar de uitgang van de chip gestuurd. Terwijl de elementen van de even beeldlijnen gedurende een periode van 20 msec worden belicht, wordt de lading van de oneven beeldlijnen naar de uitgang getransporteerd. In de volgende periode van 20 msec worden de elementen van de oneven beeldlijnen belicht en wordt de lading van de even beeldlijnen naar de uitgang getransporteerd. De frequentie waarmee de (volledige) beelden

naar de uitgang worden gestuurd is gelijk aan 25 Hz.
Aan de uitgang wordt de beeldinformatie (lading) omgezet in een analoog videosignaal dat voldoet aan de CCIR-standaard (zie figuur a.2). De 'negatieve' pulsen in dit signaal markeren het begin van de beeldlijnen. Deze pulsen worden de *horizontale synchronisatiepulsen* genoemd. Het transporteren van de lading van één beeldlijn naar de uitgang neemt 64 µsec in beslag.

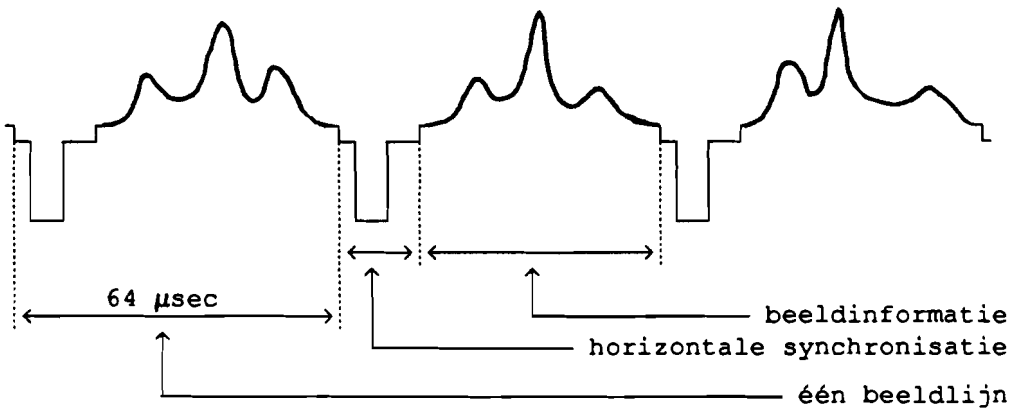
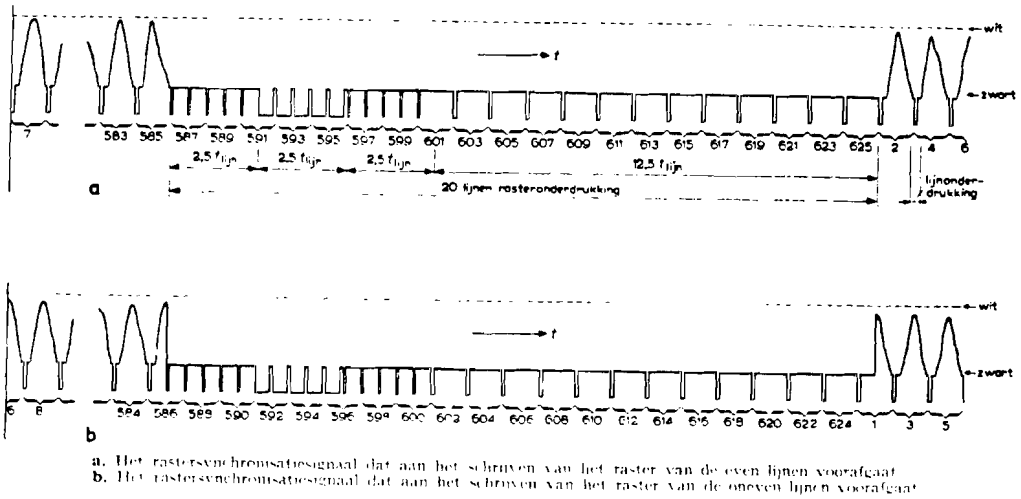


fig a.2: Drie beeldlijnen in een videosignaal.

Zoals reeds vermeld wordt eerst de beeldinformatie van de oneven lijnen naar de uitgang getransporteerd en daarna de beeldinformatie van de even beeldlijnen. Het begin van elk der halfbeelden wordt aangegeven d.m.v. een serie pulsjes zoals weergegeven in figuur a.3. Deze twee series pulsjes worden de *verticale synchronisatie* genoemd.



a. Het rastersynchronisatiesignaal dat aan het schrijven van het raster van de even lijnen voorafgaat
b. Het rastersynchronisatiesignaal dat aan het schrijven van het raster van de oneven lijnen voorafgaat

fig. a.3: Videosignaal met verticale synchronisatie.

De camera van HTH heeft twee ingangen, n.l.:

- een *clock*-ingang
- en een *reset*-ingang.

De clockingang is nodig wanneer men het videosignaal pixelsynchroon wil bemonsteren. Op de clock-ingang wordt een pulssignaal aangesloten met een frequentie van 11.25 MHz en ditzelfde pulssignaal gebruikt men de bemonstering van het videosignaal te synchroniseren.

Door een resetpuls te geven op de reset-ingang van de camera kan tevens het moment waarop de camera het eerste halfbeeld naar de uitgang stuurt worden gesynchroniseerd. Nadat de resetpuls is gegeven duurt het 20 msec voordat het eerste halfbeeld op de uitgang van de camera verschijnt. Gedurende deze 20 msec wordt het ladingspatroon in de lichtgevoelige cellen op de CCD-chip opgebouwd.

Appendix B: General purpose beeldverwerkingssysteem: Image Processor Series 150/151

Imaging Techology levert een programmeerbaar beeldverwerkingssysteem - n.l. de Image Processor Series 150/151 - waarop eenvoudige wijze verschillende beeldverwerkingstechieken kunnen worden ontwikkeld en getest.

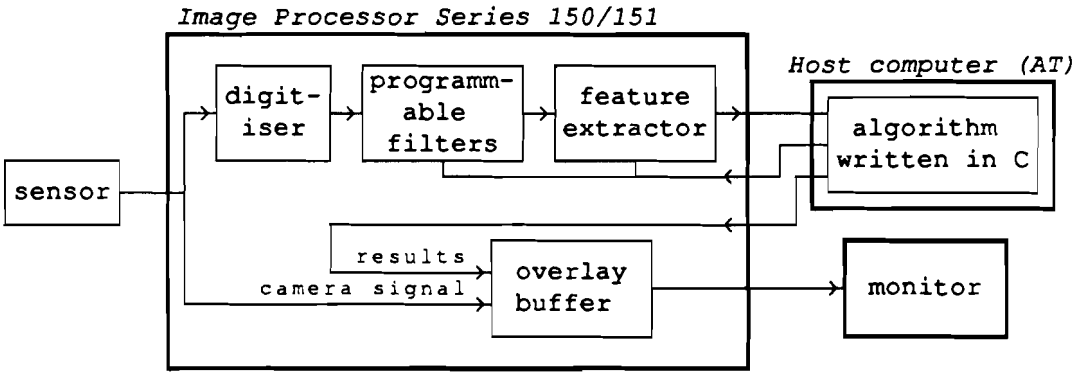


fig. b.1: Programmeerbaar beeldverwerkingssysteem Image Processor 150/151.

Een schema van het systeem is weergegeven in figuur b.1. Het systeem bevat ondermeer een digitiser, die een analoog videosignaal, afkomstig van een standaard videocamera, omzet in een gedigitaliseerd beeld. Het systeem bevat verder een aantal programmeerbare digitale filters, zowel 1- als 2-dimensionale filters. Deze filters bewerken het gedigitaliseerde beeld in realtime. De resultaten van de filteroperaties komt terecht in de zogenaamde feature extractor. Deze kunnen vandaaruit worden overgebracht naar een host computer - bijvoorbeeld een IBM AT (MSDOS) of een SUN-computer (UNIX) - die op het beeldverwerkingssysteem is aangesloten.
Voor de hostcomputer kan software worden ontwikkeld om de reeds voor bewerkte beeldinformatie verder te analyseren. Imaging Technology ondersteunt de communicatie tussen het systeem en de hostcomputer met speciale software. Deze software kan worden aangeroepen vanuit programma's die geschreven zijn in de programmameertaal *C* (Microsoft C-compiler).

De communicatiesoftware maakt het ondermeer mogelijk dat:

- vanuit de host de 1- en 2-dimensionale filters van het beeld-verwerkingssysteem kunnen worden geprogrammeerd.
- beeldinformatie vanuit het beeldverwerkingssysteem naar de host kan worden overgebracht.

Nadat de beeldinformatie door beeldverwerkingsprogrammatuur op de hostcomputer is verwerkt kunnen de resultaten hiervan, samen met het originele camerabeeld, zichtbaar worden gemaakt op een monitor.

Appendix C: Memory map ADSP-2100

The ADSP-2100 micro processor has a seperated bus structure for the program memory and the data memory (figure c.1). The advantage of this architecture is that at the same time data can be accessed in the data memory space and a new instruction can be fetched from the program memory.

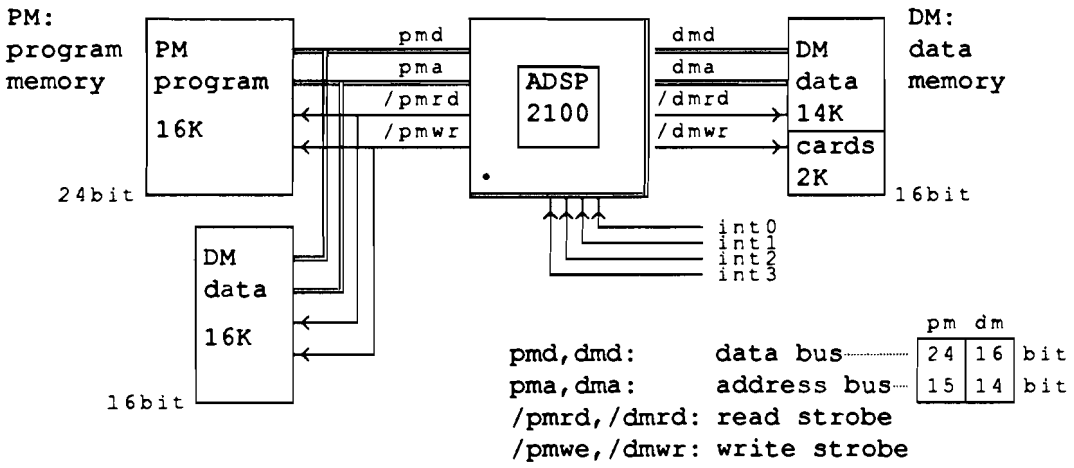


fig. c.1: Harvard architecture: the program memory and the data memory both have their own address and data bus.

The address bus of the program memory is 15 bit wide, the address bus of the data memory is 14 bit wide. The data bus of the program memory is 24 bit wide, the data bus of the data memory is 16 bit wide.

The program memory and the data memory are configurated as follows:

Program Memory

RAM:	program code, address space:	h#0000-h#3fff
RAM:	data, address space:	h#4000-h#7fff

Data Memory

RAM:	data, address space:	h#0000-h#37ff
CARDS(0-7):	registers mapped in address space:	h#3800-h#3fff

Note that the last block of 2K in the data memory is reserved for a number of registers (also called ports). This block is seperated in 8 pages that can be addressed using the 9th, the 10th and the 11th bit of the data memory address.
For every application board that will be connected to the data memory address bus a page (0-7) has to be assigned. Table c.1 shows the pages that already have been assigned.
When a register is addressed a acknowledge pulse (active low) has to be send to the ADSP via the acknowledge line on the bus backplane that is specified in the item Acknowledge. This construction allows to use components with a low access time.

Table c.1: Configuration of data memory block h#3800-h#3fff

Card id.	Data Memory Adress		r/w	Acknow-ledge	Register Name
	Binary	Hexa*			
0 monitor	1110001xxxxxxx	h#38FF	w	ACK0	PHASEREG
1 card1	111001xxxxxxxx	h#39xx	-	ACK1	-
2 card2	111010xxxxxxxx	h#3Axx	-	ACK2	-
3 card3	111011xxxxxxxx	h#3Bxx	-	ACK3	-
4 card4	111100xxxxxxxx	h#3Cxx	-	ACK4	-
5 video	111101xxxxxxxx	h#3Dxx	-	ACK5	-
6 fifo	111110111xxxx0	h#3EFE	r	ACK6	FIFOFLAGS
	111110111xxxx1	h#3EFF	r/w	ACK6	FIFODATA
7 coordinate generator	1111111xxxxxx0	h#3FFE	r	ACK7	XREG
	1111111xxxxxx1	h#3FFF	r	ACK7	YREG

*) Not all address bits are considered when a register on a card is selected. The bits that are not considered are the so called "don't care" bits, represented as an "x" in the binary representation of the address.
To address a register in the ADSP-2100 software a hexa-deciamal representation has to be used. To get one unique hexadecimal address for each register the arrangement has been made that don't care bits are substituted by a logical '1'.

Appendix D: Coordinate Generator

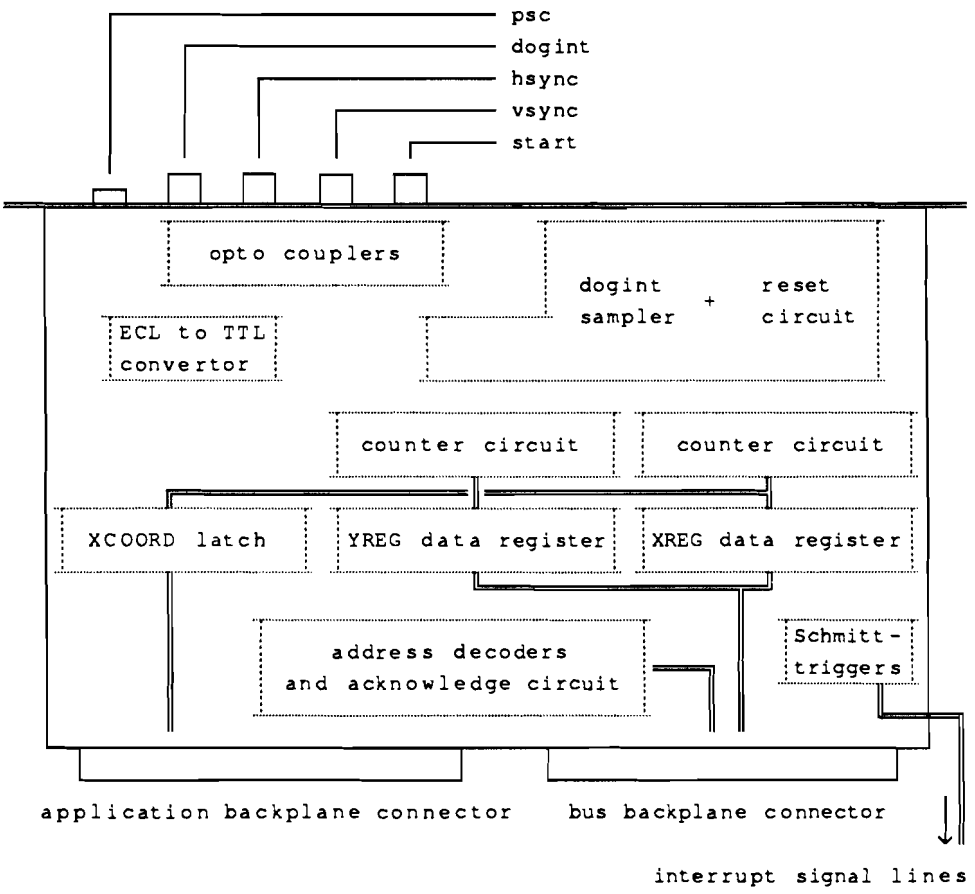


fig. d.1: Layout coordinate generator card.

The coordinate generator is implemented in hardware on a printed circuit board. The layout of the component side of this board is shown in figure d.1.
The front panel has five input connectors which are listed in tabel d.1. All input signals, except the psc, are electrically isolated using opto coupler circuits (5N135 for low frequency and 5N137 for high frequency signals). Decoupling is needed to avoid interference due to ground loops.

Table d.1: Front panel coordinate generator card.

Input		Source	Connector
psc	pixel sample clock	psc generator	twinax, ECL
dogint	interrupt pulses	pulse width filter	BNC, TTL
hsync	horizontal synchron.	sync splitter	BNC, TTL
vsync	vertical synchron.	sync splitter	BNC, TTL
start	start pulse	timing module	BNC, TTL

The card is connected to the ADSP-2100 bus structure via the busback-plane connector. The configuration of this connector is given in appendix G. This configuration is different from the configuration of the prototyping connector on the ADSP-2100 evaluation board. Many pins on the prototyping connector are redundant. Consequently the bus adaptor addapts the 96-pin prototyping connector to the 64-pin bus backplane connector. The advantage is that a standard (i.e. cheap) 64-pin backplane can be used for the bus backplane. A bus backplane is needed to add more application cards to the bus structure of the ADSP-2100. In table c.1 (appendix C) these application cards are listed. For the configuration of the 96 pin prototyping connector on the ADSP-2100 evaluation board refer to appendix F. For the configuration of the 64-pin bus backplane refer to appendix G.

The registers XREG and YREG can be accessed by the ADSP-2100 via the bus backplane connector. The addresses of these registers (XREG: h#3ffe, YREG: h#3fff) are decoded by the address decoders and an acknowledgement signal is generated.
The decoder circuit is made using high speed logic. The components shown in the circuit in appendix E belong to the TTL Advanced Shotkey family. The maximum time delay between address valid and decoder output valid is 9.5 nsec. This is fast enough for this application.

The distribution may be implemented in hardware as well. Distribution means that the coordinates that are generated by the coordinate generator should be distributed over a number of vertical areas. On the distributor card the x-coordinate of a pixel is compared with the boudaries of the different areas. The comparision can be implemented in fast logic. The coordinate generator offers x-coordinate to the distributor card via the XCOORD lacth and the application backplane. For the configuration of the application backplane refer to appendix H.

Appendix E: Coordinate generator, electric design

This appendix showns the design of the coordinate generator card on component level.
The coordinate generator module is seperated in submodules. The first sheet shows an overview of all submodules on the card. The next sheets show the submodule in detail:

sheet 2

The input signals *start, hsync, vsync* and *dogint* are all TTL signals. These signals are high active. The input signals *psc ph1* and *psc ph2* are the two phases of the ECL signal.

sheet 3

The *reset* and *non-reset (/reset)* are generated using the vertical synchronisation pulse.

sheet 4

This sheet shows two 12-bit counter circuits, made of 4-bit binary counters (74ALS161). The x-coordinate of the current pixel in the video frame is shown on the output of the upper counter circuit. The y-coordinate of this pixel is shown on the output of the lower counter.
The counter circuits are controlled by the *psc, hsync, reset* and */reset* signals.

sheet 5

The dogint input is active low. The dogint pulses have a pulse width of about 50 nsec. These pulses are lengthened by the dogint sampler to pulses with a pulse width of 180 till 270 nsec (between two or three times the period of the psc signal, which has a period of 90 nsec).
After $2^1/2$ periodes of the pixel sample clock, after the D-flipflop U45B changes from low to high, the *clockreg* signal (clock registers) becomes active high, during one period of the pixel sample clock.

sheet 6

The decoder contols the OC-inputs (output control) of the data registers XREG and YREG. If these registers are addressed a acknowledge pulse has to be send to the evaluation board. The acknowledge output *(ack7)* ulse is active low.

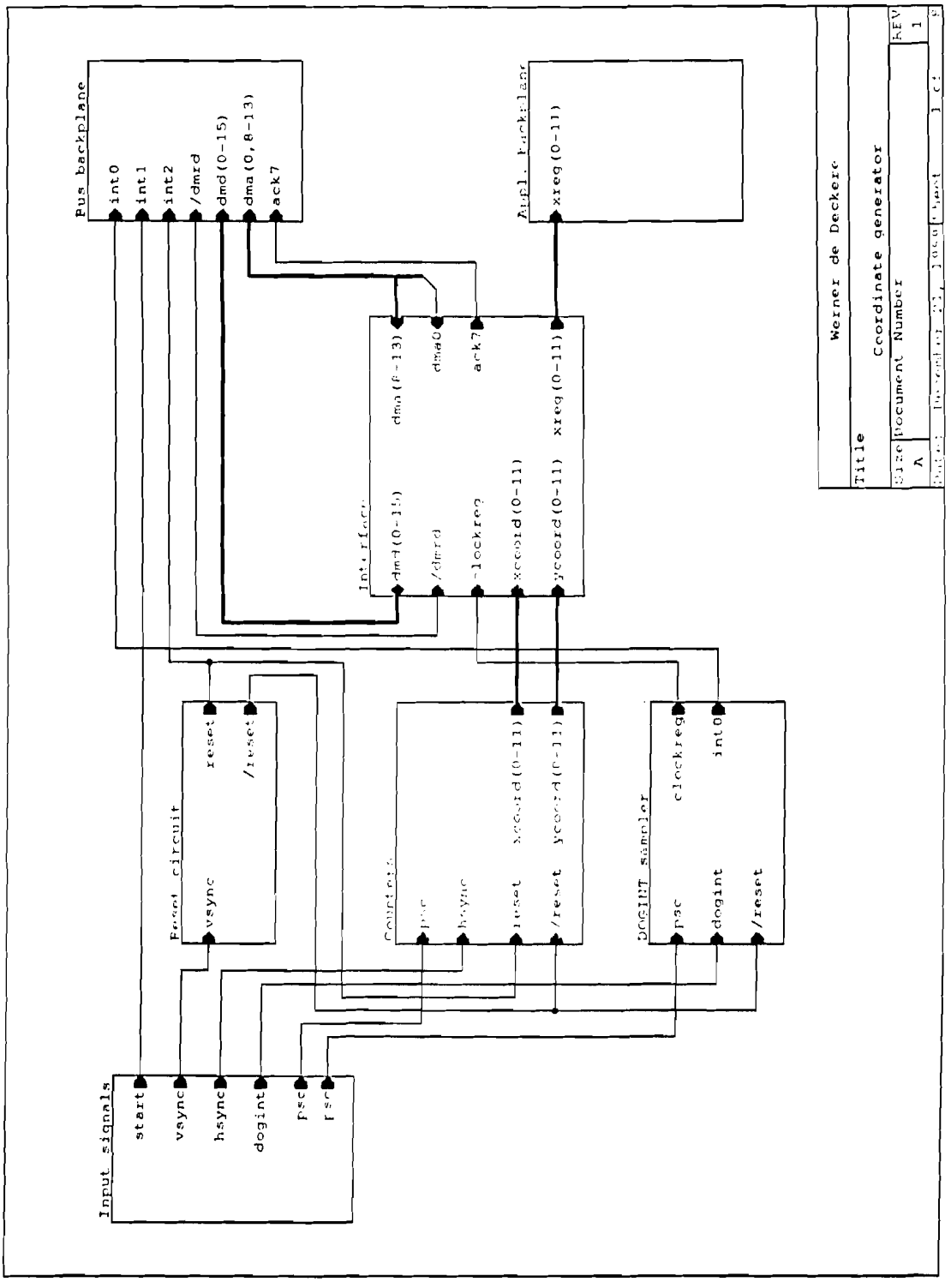


fig. e.1: General overview.

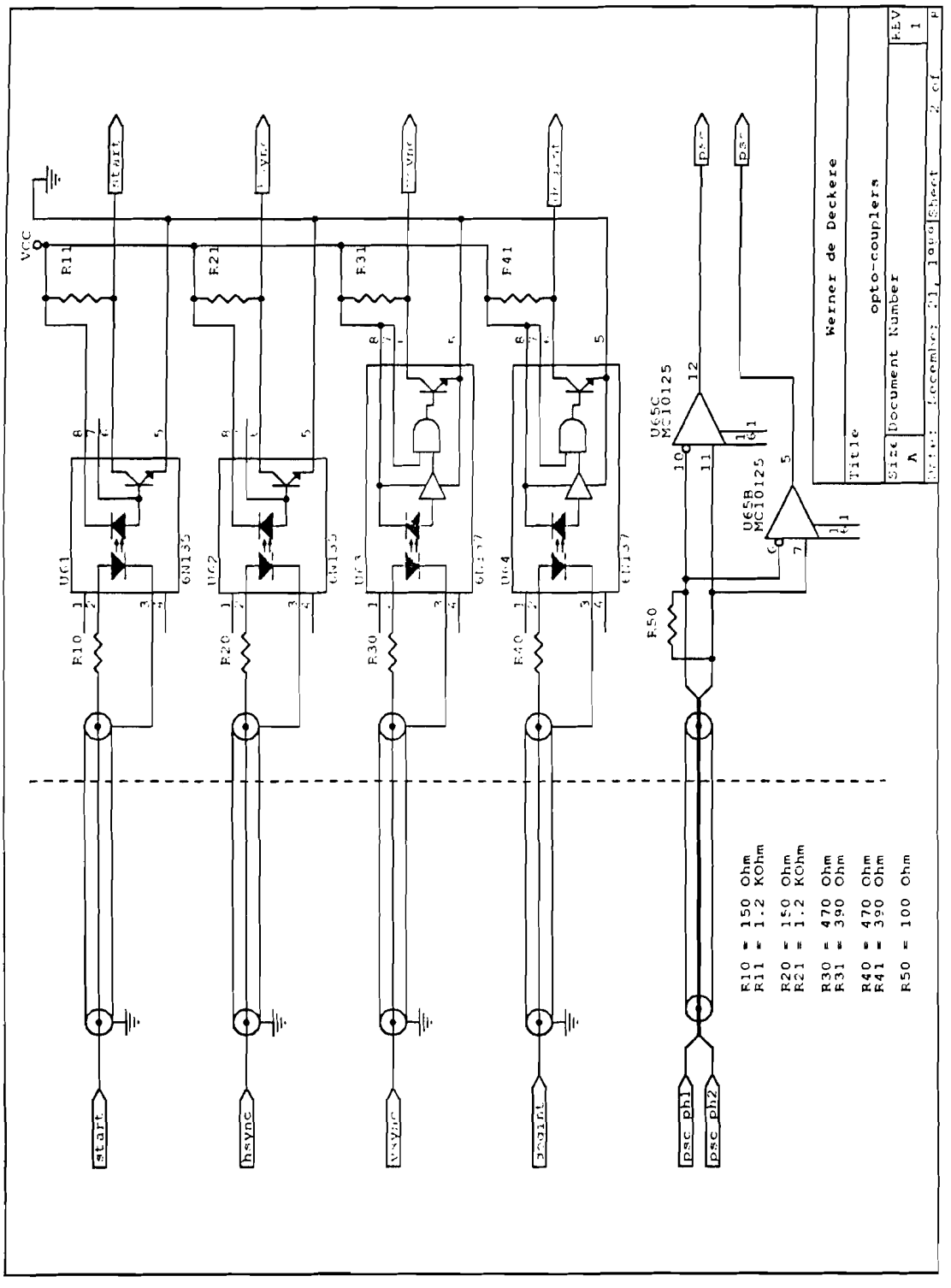


fig. e.2: Opto-couplers.

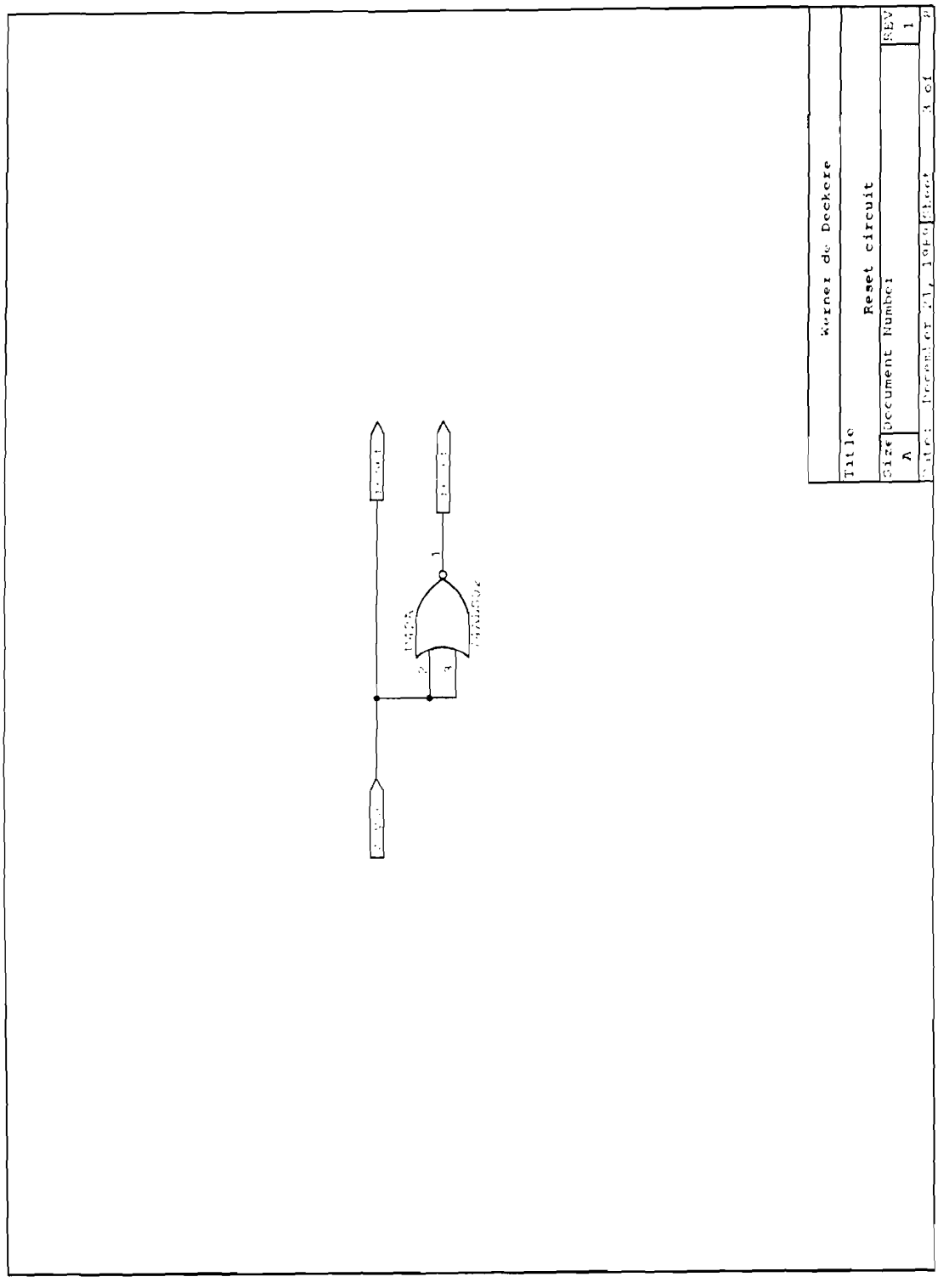


fig e.3: Reset circuit.

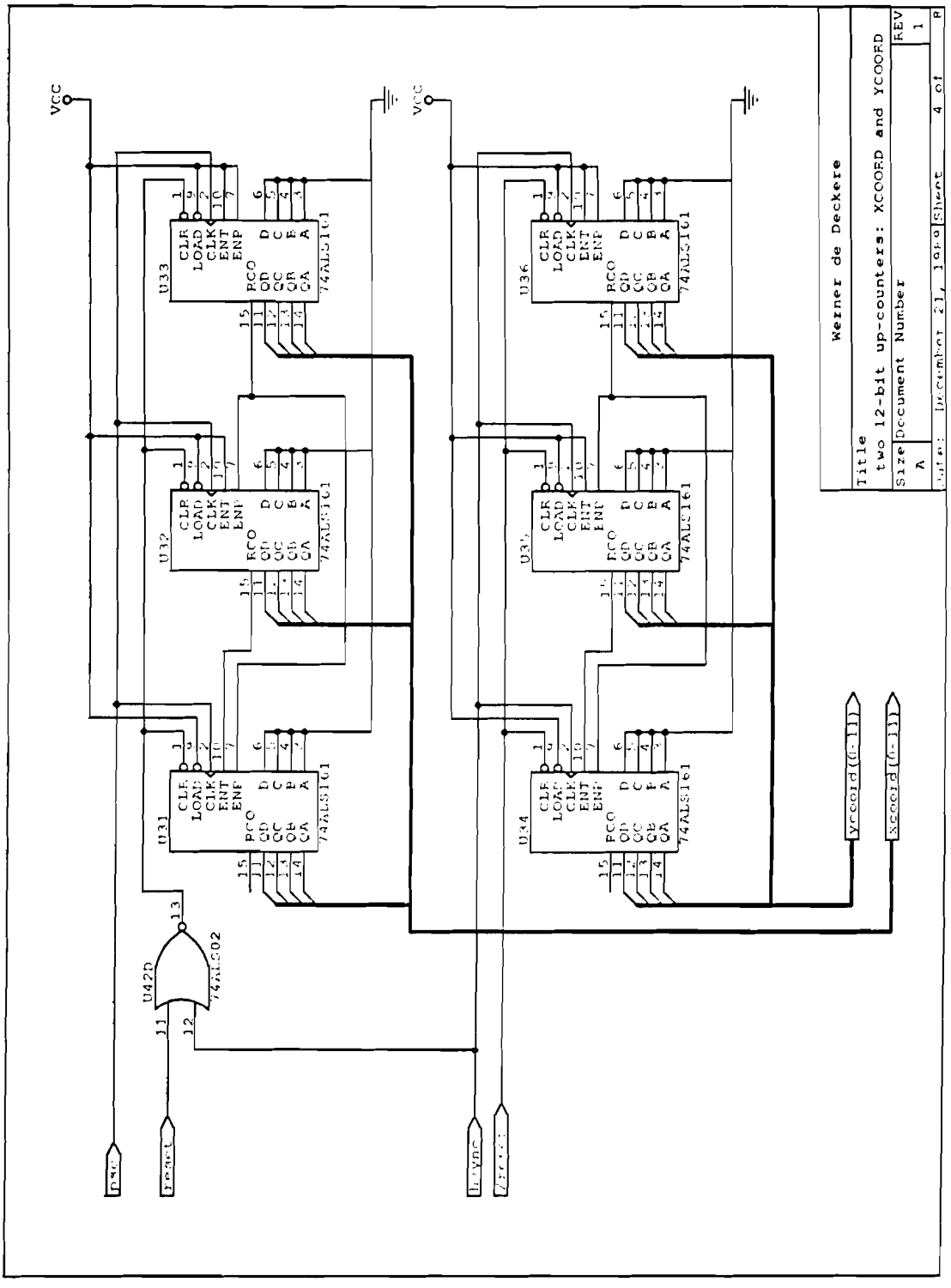


fig. e.4: Two 12-bit counter circuits.

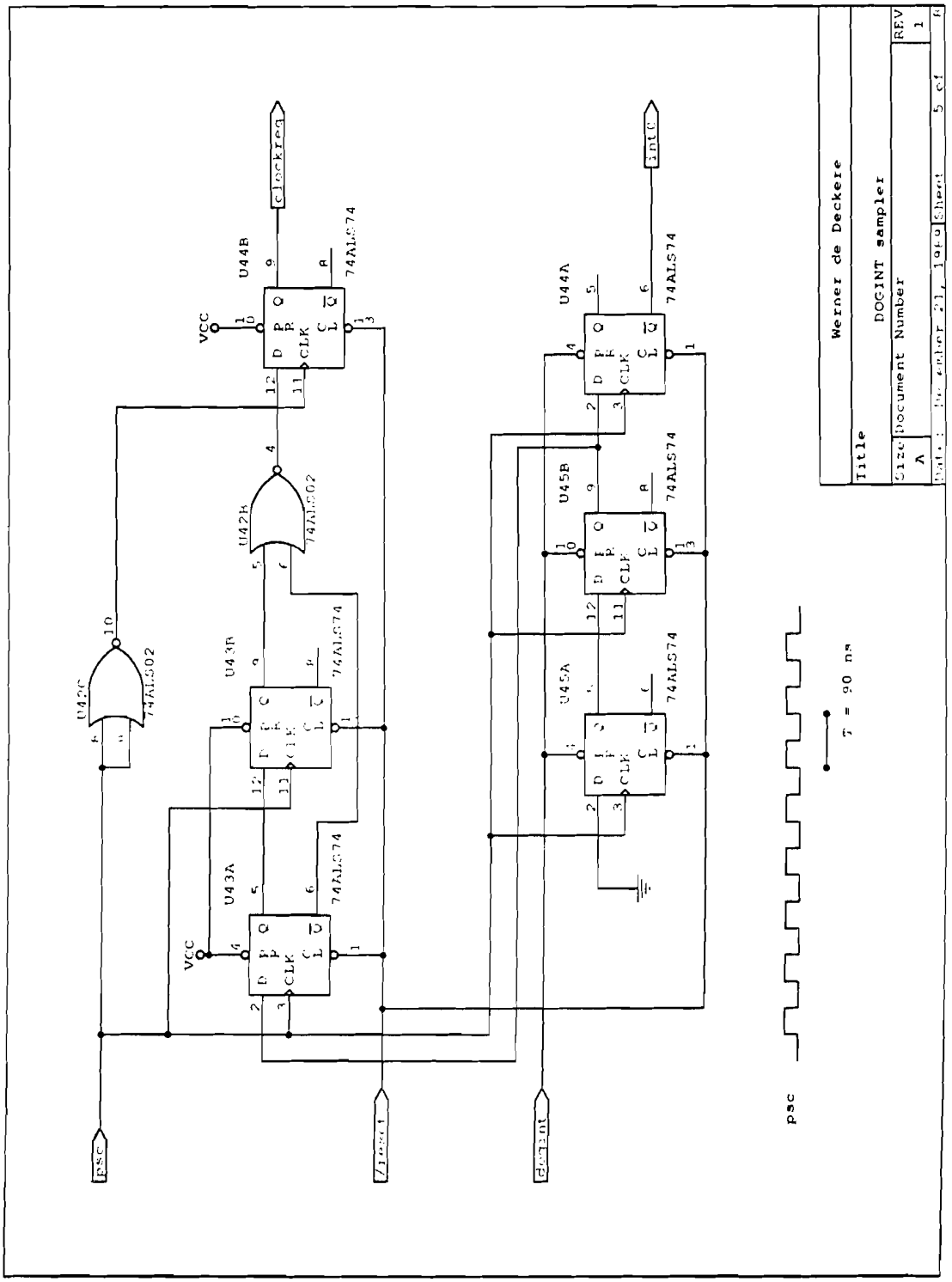


fig. e.5: Dogint sampler.

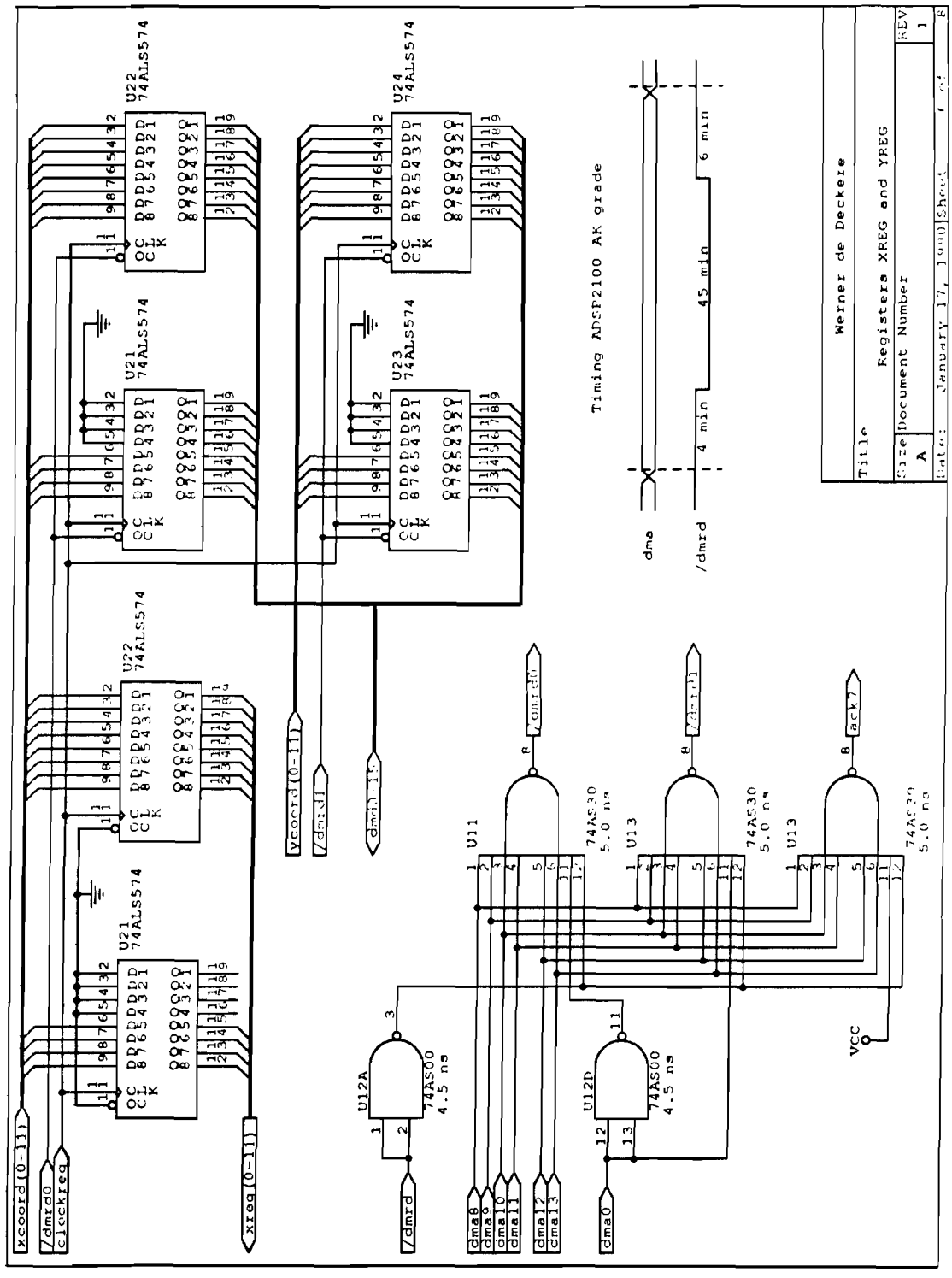


fig. e.6: Registers XREG and YREG.

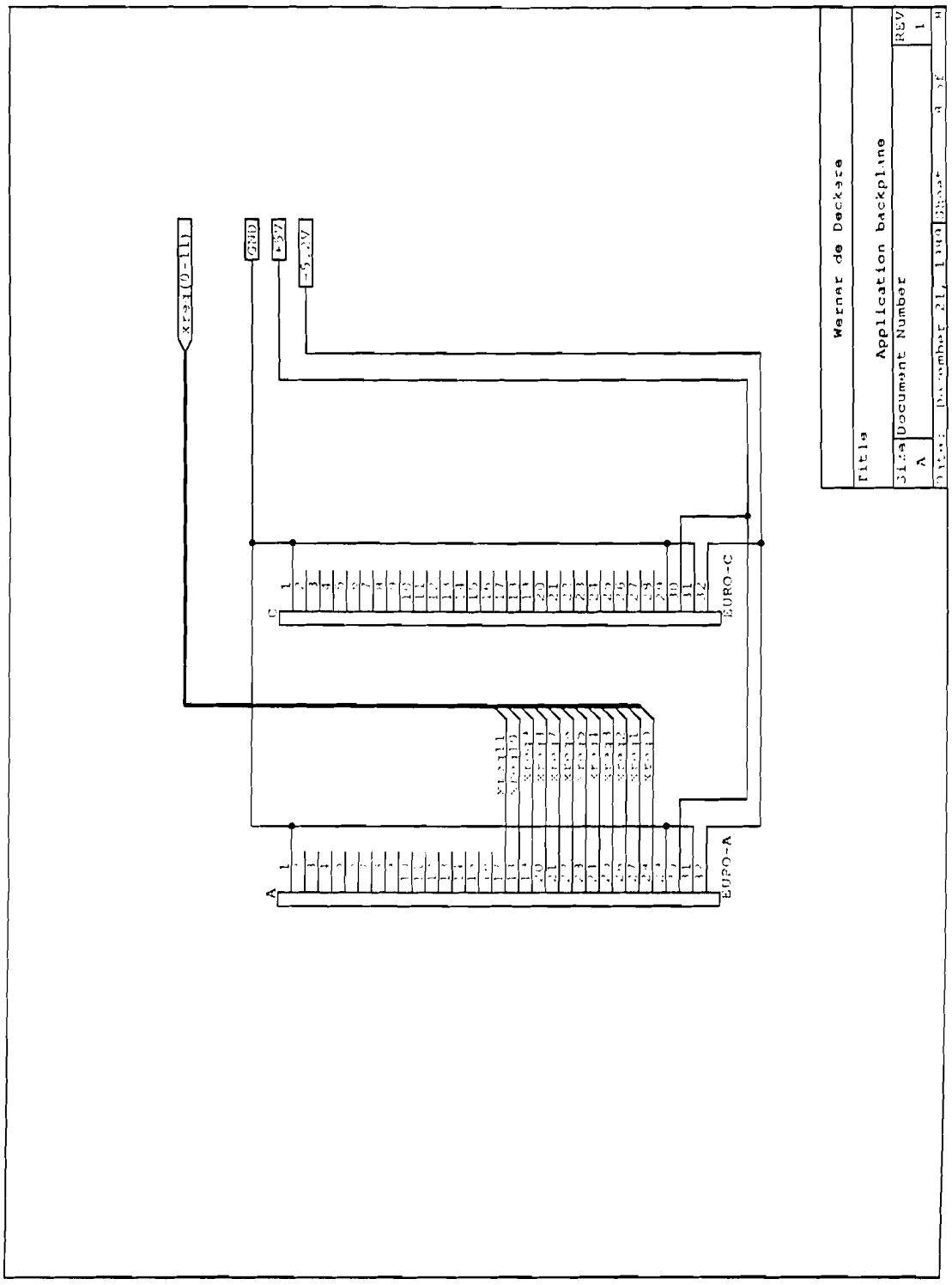


fig. e.7: Application backplane.

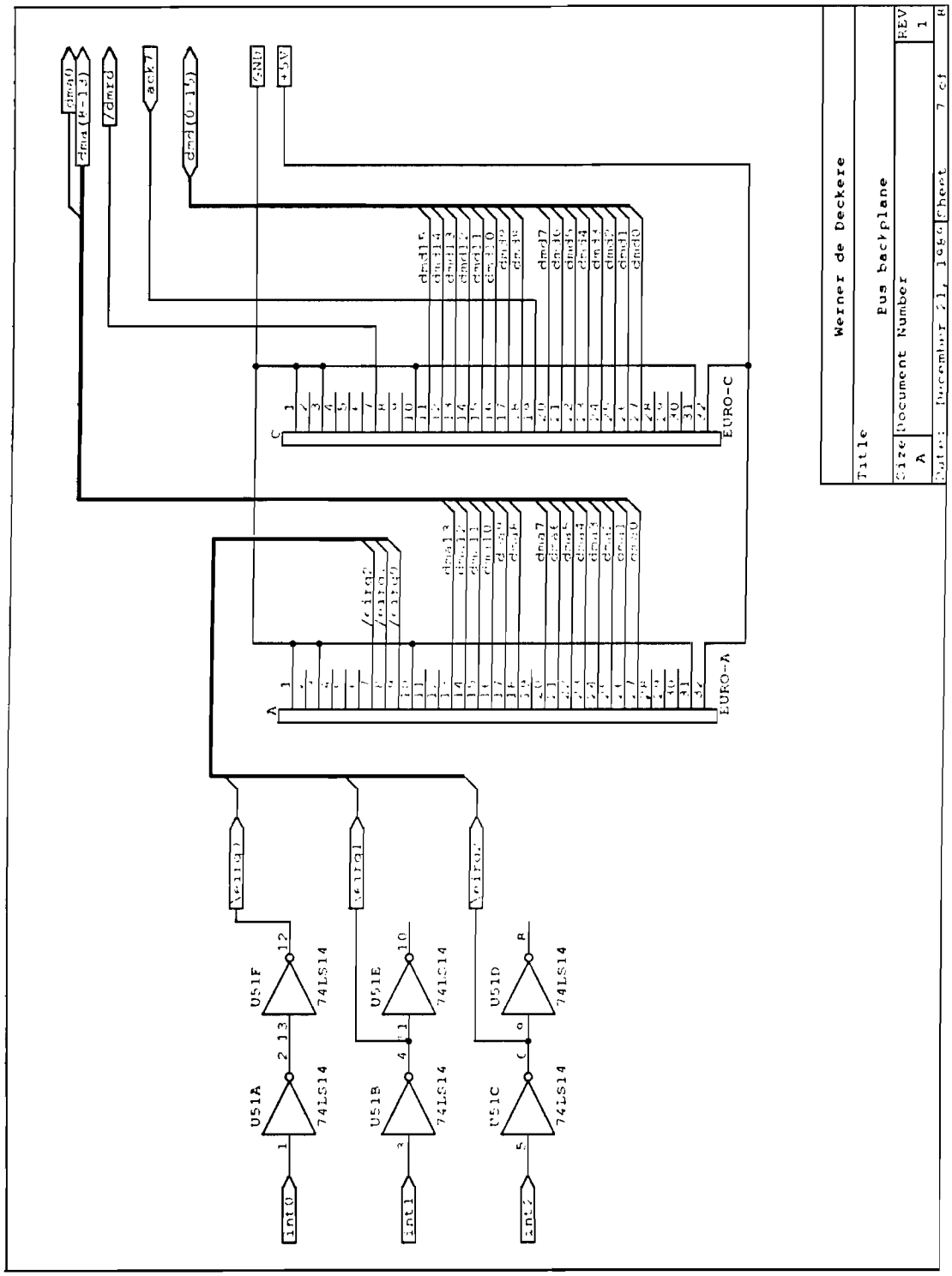
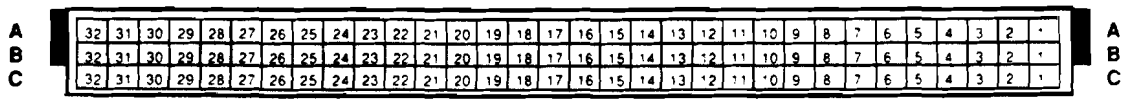


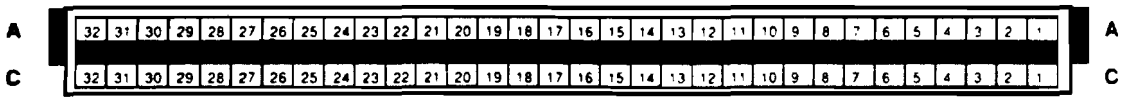
fig. e.8: Bus backplane.

Appendix F: Prototyping connector ADSP-2100



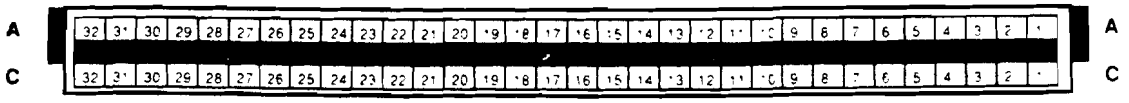
	A	B	C
1	+12V	+12V	+12V
2	Analog GND	Analog GND	Analog GND
3	-12V	-12V	-12V
4	Analog GND	Analog GND	Analog GND
5	Digital GND	Digital GND	Digital GND
6	/EIRQ3	EDMACK	CLOCKOUT
7	/EIRQ2	/DMRD	/TRAP
8	/EIRQ1	/DMWR	/THALT
9	/EIRQ0	/DMS	/RESETOUT
10	Digital GND	Digital GND	Digital GND
11	/EBR	DMD15	N/C
12	/BG	DMD14	N/C
13	DMA13	DMD13	N/C
14	DMA12	DMD12	N/C
15	DMA11	DMD11	N/C
16	DMA10	DMD10	N/C
17	DMA9	DMD9	N/C
18	DMA8	DMD8	N/C
19	Digital GND	Digital GND	Digital GND
20	DMA7	DMD7	/ECE8
21	DMA6	DMD6	/ECE7
22	DMA5	DMD5	/ECE6
23	DMA4	DMD4	/ECE5
24	DMA3	DMD3	/ECE4
25	DMA2	DMD2	/ECE3
26	DMA1	DMD1	/ECE2
27	DMA0	DMD0	/ECE1
28	Digital GND	Digital GND	Digital GND
29	+5V Digital	+5V Digital	+5V Digital
30	Digital GND	Digital GND	Digital GND
31	+5V Digital	+5V Digital	+5V Digital
32	Digital GND	Digital GND	Digital GND

Appendix G: Bus backplane connector



	A	C
1	GND	GND
2	+12V	-12V
3	N/C	N/C
4	TRAP	CLOCKOUT
5	/THALT	/RESETOUT
6	/EIRQ3	EDMACK
7	/EIRQ2	/DMRD
8	/EIRQ1	/DMWR
9	/EIRQ0	/DMS
10	N/C	N/C
11	/EBR	DMD15
12	/BG	DMD14
13	DMA13	DMD13
14	DMA12	DMD12
15	DMA11	DMD11
16	DMA10	DMD10
17	DMA9	DMD9
18	DMA8	DMD8
19	/ACK3	/ACK8
20	DMA7	DMD7
21	DMA6	DMD6
22	DMA5	DMD5
23	DMA4	DMD4
24	DMA3	DMD3
25	DMA2	DMD2
26	DMA1	DMD1
27	DMA0	DMD0
28	/ACK2	/ACK6
29	/ACK1	/ACK5
30	/ACK0	/ACK4
31	N/C	N/C
32	+5V Digital	+5V Digital

Appendix H: Application backplane connector



	A	C
1	GND	GND
2	N/C	N/C
3	N/C	N/C
4	N/C	N/C
5	N/C	N/C
6	N/C	N/C
7	N/C	N/C
8	N/C	N/C
9	N/C	N/C
10	N/C	N/C
11	N/C	N/C
12	N/C	N/C
13	N/C	N/C
14	N/C	N/C
15	N/C	N/C
16	N/C	N/C
17	N/C	XREG11
18	N/C	XREG10
19	N/C	XREG9
20	N/C	XREG8
21	N/C	XREG7
22	N/C	XREG6
23	N/C	XREG5
24	N/C	XREG4
25	N/C	XREG3
26	N/C	XREG2
27	N/C	XREG1
28	N/C	XREG0
29	N/C	GND
30	-5.2V Analog	-5.2V Analog
31	GND	GND
32	+5V Digital	+5V Digital

Appendix I: ADSP-2100 Cross-software

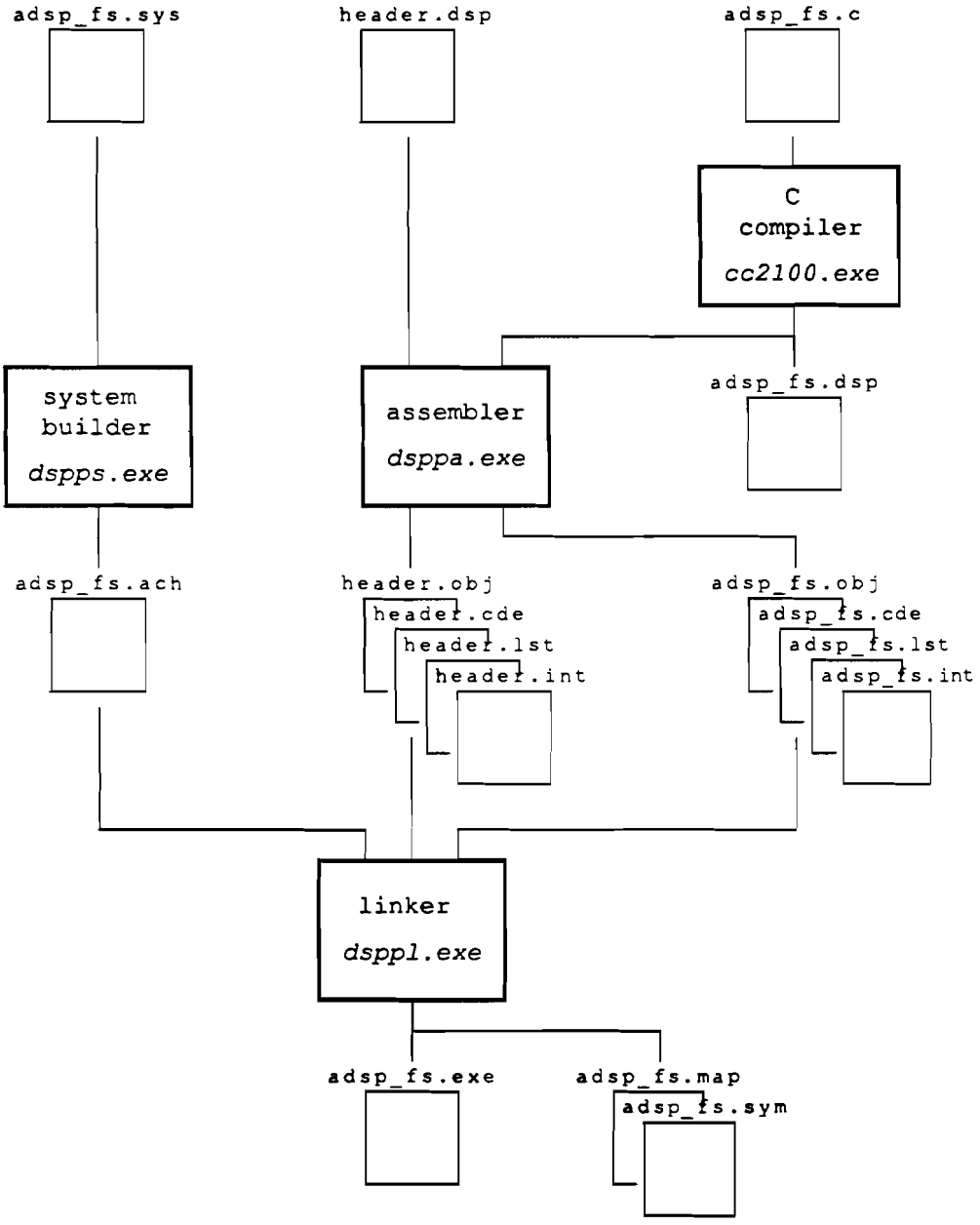


fig. i.1: Cross software for compiling source files.

The cross-software system has six modules: System Builder, Assembler, C-compiler, Linker, Simulator and PROM Splitter.

The System Builder converts the *system file (adsp_fs.sys)* to the *architecture file (adsp_fs.ach)*. In the system file the amount of RAM and ROM available and the allocations of program memory, data memory and I/O ports are specified. The Linker needs the architecture file to allocate the program code and variables in the ADSP's address space.

The image processing software consists of two modules. The first module is the so called *header (header.dsp)* and is written in the ADSP-2100 assembly code. This module contains stack initialisations and the code of the interrupt service routines. The source file is assembled to an object file *(header.obj)*.
The second module *(adsp__fs.c)* contains the image processing algorithms written in the C programming language (ANSI standard). The C-compiler produces ADSP-2100 assembly code *(adsp_fs.dsp)* and directly invokes the Assembler to generate an object file *(adsp_fs.obj)*.

The linker links the object files produced by the Assembler. It maps the linked code and data output to the target system hardware. This is done using the the architecture file *(adsp_fs.ach)*. The linker generates a list of machine instructions for the ADSP-2100, listed in *adsp_fs.exe*.

For the Simulator and the PROM Splitter refer to the Cross software manual [2].

Use the following command to invoke the System Builder:

```
dsppb adsp_fs.sys
```

To invoke the Assembler, the C-compiler and the Linker:

```
dsppa -c header
cc2100 adsp_fs -m
dsppl header adsp_fs -a adsp_fs -c -e adsp_fs -g -x
```

To use to cross software in a more user friendly way, a batch file has been made. This batch file, called *m.bat*, is listed in appendix K.
To invoke this batch file type:

m [return] , to assemble the header file, compile the C-program and link the object files
m a [return], to assemble the header file and link the object files
m c [return], to compile the C-program and link the object files

Note: Always remove resident programs from memory before invoking one of the cross-software modules. If not then sometimes the modules don't accept its source files.

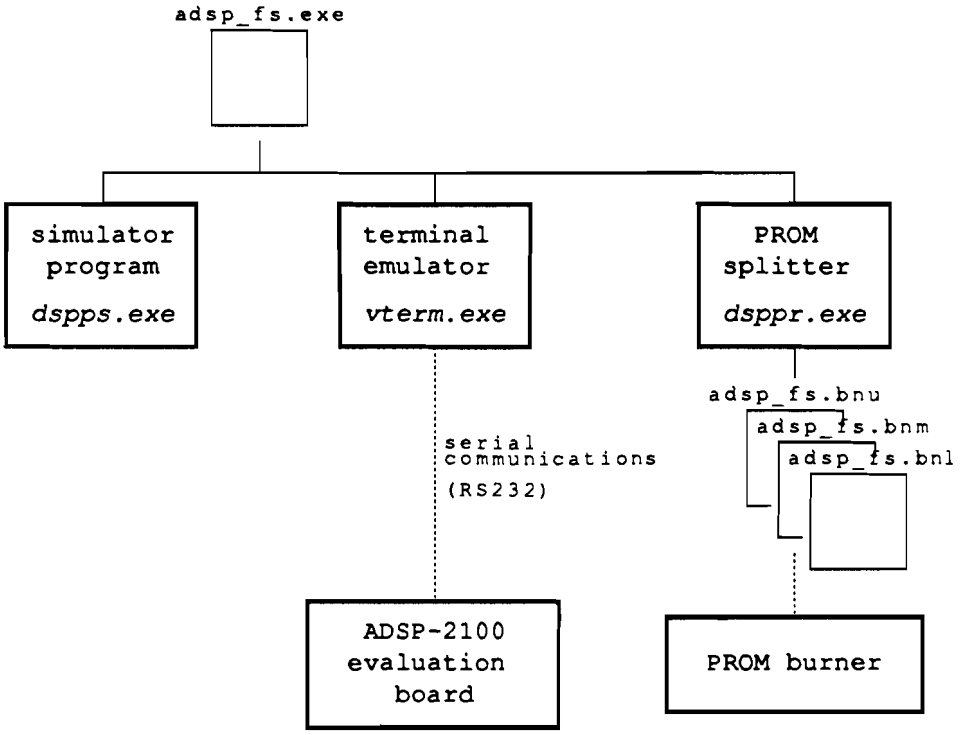


fig. i.2: Simulation, evaluation board and PROM burner

Now that the sources are translated to a list of instructions for the digital signal processor, this instructions have to be downloaded in the program memory of the signal processor on the evaluation board via a serial communication line (RS232). On the host computer a terminal emulator program is used to communicate with the evaluation board. To invoke the VTERM terminal emulation program, type:

vterm [return]

Use:

F5 to show the setup screen of the emulator program.

Change the contents of the following items in the column on the right into:

```
baudrate          = 9600
data bits/parity  = 8/none
stopbits          = 1
auto xon/xoff     = off
```

Use:

(Esc)	to leave the setup screen.

Now press the reset button on the evaluation board. On the terminal all kind of information is shown showing the status of the ADSP-2100 digital signal processor on the evaluation board. Type in the following commands:

ibm [return]	to tell the evaluation board that we are working on a ibm (compatible) computer.
(Ctrl-)D	to go to the so called transparrent mode (the information on the screen disappears).
(Alt-)S	to show the pop-up screen to specify the file which has to be sent and protocol which is used for transmitting the data.
adsp_fs.exe	enter the name of the file to be sent and select ASCII TEXT on the next line, being the the name of the protocol.
(Alt)-S	to start sending the file.

After the file has been sent, press:

(Ctrl-)D	to get back screen showing the status information of the ADSP-2100

Now you can test the program that has been downloaded in the program memory of the digital signal processor. Refer to the evaluation board manual [3] for the commands you need to do this.

Note:

To stop the terminal emulation program while the program remains resident in memory, press:

Shift-Left and **Shift-Right** at the same time.

To invoke the resident program, press:

Shift-Left and **Shift-Right** at the same time.

To stop the terminal emulation program and remove the program from memory, press:

(Ctrl-)Break

Appendix J: Listing image processing software: general purose Image Processor Series 150/151

The contents of this appendix are:

fsmod3.c	Module 3 of the seam finding software for the Image Processor Series 150/151.

For the modules fsmod0.c, fsmod1.c and fsmod2.c refer to the *pattern recognition software diskette* (see also appendix L) in the software librarary managed by:

Will Hendrix,
Measurement and Controll department (ER)
Robot vision group
faculty of Electrotechnical Engineering
Eindhoven University of Technology

```
/*-------------------------------------------------------------------------*

   Source file name: fsmod3.c    (algorithm for finding overlap seams)
   Author          : Werner de Deckere
   Date            : Januari 1990

   Eindhoven University of Technology
   Faculty of Electrotechnical Engineering
   Department Measurement and Control
   P.O. Box 513, 5600 MB Eindhoven, The Netherlands

 *-------------------------------------------------------------------------*

   This source file contains the algorithms to find an overlap seam in a
   piece of work. The C programming language is used to describe the
   algorithms.
   To compile the sourcefile and link the object file with the object
   files of the following modules:

      fsmod0.c
      fsmod1.c
      fsmod2.c

   use the MicroSoft make utility:

      make fs

   (How the compilation and the linking has to be done is specified in
    the file named: fs)


   For the description of the subroutines, defined in this sourcefile,
   refer to the sourcefile: adsp_fs.c

 *-------------------------------------------------------------------------*/
#include <itex150.h>
#include <regop.h>
#include <stdio.h>
#include <screen.h>


/* definition of some constants */
#define FIELD_SIZE                      1024
#define PIXELS_PER_SCANLINE              604
#define NUMB_SCANLINES                   256
#define MAX_NUMB_AREAS                     5
#define Y_MAX                            350
#define Y_COMP                             4

/* field data structure */
struct field
{
    int x,y;
};
struct field *fldptr, *fldptr_max;
struct field fld[FIELD_SIZE];

int phase, ready;
struct field *fldp;
struct field *distr_ptr;

counter[MAX_NUMB_AREAS], x_prev[MAX_NUMB_AREAS];

int stripe[MAX_NUMB_AREAS][Y_MAX];
int convol[MAX_NUMB_AREAS][Y_MAX];
int *ptr, *help_ptr;
int length[MAX_NUMB_AREAS],
    updated[MAX_NUMB_AREAS],
    flag[MAX_NUMB_AREAS],
    x_prev[MAX_NUMB_AREAS], y_prev[MAX_NUMB_AREAS],
    max, min, y_max, state;

int numb_areas = 3; /*MAX_NUMB_AREAS; */


struct seam_elem_record {float x,y,z;};
struct seam_elem_record seam_elem[MAX_NUMB_AREAS][16], *seam_elem_ptr;
int numb_seam_elem[MAX_NUMB_AREAS];

/* menu-variables */
int displacement    =                    40;
int diff_x_min0     =                    12;
long sum1_max       =                    20;
long sum2_min       =                    40;
long sum2_max       =                   300;
int diff_x_min1     =                   150;
int diff_x_max1     =                   500;
int diff_x_half     =                     5;
int diff_y_max      =                    16;
int direction       =                     1;

int stripe_x[MAX_NUMB_AREAS] = {180,285,400,400,500};
int area_bound[MAX_NUMB_AREAS];

float origin_x, origin_y, origin_z;
float x_adjust = 1;
float y_adjust = 1;
float z_adjust[MAX_NUMB_AREAS] = {1,1,1,1,1};
/* z_adjust: 1 devided bij tangus of alfa */

int x_coord, y_coord;
int count, bound, area, *area_ptr, l, h;
int i, j;

int dog_kernel[16] = { -1, -3, -3, -1,  0,  1,  3,  4,
                        4,  3,  1,  0, -1, -3, -3, -1};
long sum;
int sum1, sum2;

int x, y, helpvar, diff;
int k;

int print_on = 1;


/*-------------------------------------------------------------------------*
   Arrays with data for the LUTs. When videodata is sent to the screen
   and one of the LUTs 1-12 is selected this videodata will be showed
   in a color specified by the r, g and b-arrays.
   LUT1  red      LUT2  orange   LUT3  yellow   LUT4
   LUT5  green    LUT6           LUT7  cyan     LUT8
   LUT9  blue     LUT10          LUT11          LUT12
 *-------------------------------------------------------------------------*/
int r[12] = {255,255,255,128,  0,  0,  0,  0,  0,128,255,255},
    g[12] = {  0,128,255,255,255,255,255,128,  0,  0,  0,  0},
    b[12] = {  0,  0,  0,  0,  0,128,255,255,255,255,255,128};



/*-------------------------------------------------------------------------*
   Some variables that are declared in other sourcefiles have to be
   specified
```

```
 *-------------------------------------------------------------------*/
extern fentry *featdata;
extern int nrfeat;
extern int demo_switch;
  /* demo_switch = 0 for real time
                 = 1 for single step
  */

extern int XCOMP2;
extern int YCOMP2;

extern show_arrow_bar();
extern int menu_select();
extern void best_fit_y(int, int*, int*);


/*-------------------------------------------------------------------*
   seam find algorithm
   construction:
     -distribution algorithm
     -linear approximation
     -parsing algorithm

 *-------------------------------------------------------------------*/



initialize_colors()
/*-------------------------------------------------------------------*
   Download the LUTs 0-12 with the data stored in the r, g and
   b-arrays.
 *-------------------------------------------------------------------*/
{
    int i;

    for(i=1; i <= 12; i++)
    {
        adi_hbanksel(i);
        adi_hgroupsel(RED); adi_clearlut(r[i-1]);
        adi_hgroupsel(GREEN); adi_clearlut(g[i-1]);
        adi_hgroupsel(BLUE); adi_clearlut(b[i-1]);
    }
}


par_menu4()
/*-------------------------------------------------------------------*
   This function is called externally in sourcefile fsmod1.c
   It is needed for changing the parameters of the algorithm described
   in this sourcefile, namely 'alg4'.
 *-------------------------------------------------------------------*/
{
    int item=1;

    while(1)
    {
        CLS
        printf("Change:\n\n");
        printf("    1 : displacement  = %d\n", displacement);
        printf("    2 : diff_x_min0   = %d\n", diff_x_min0);
        printf("    3 : sum1_max      = %d\n", sum1_max);
        printf("    4 : sum2_min      = %d\n", sum2_min);
        printf("    5 : sum2_max      = %d\n", sum2_max);
        printf("    6 : direction     : ");
        if(direction == 1) printf("right\n"); else printf("left\n");
        printf("    7 : stripe_x[0]   = %d\n", stripe_x[0]);
        printf("    8 : stripe_x[1]   = %d\n", stripe_x[1]);
        printf("    9 : stripe_x[2]   = %d\n", stripe_x[2]);
        printf("   10 : stripe_x[3]   = %d\n", stripe_x[3]);
        printf("   11 : stripe_x[4]   = %d\n", stripe_x[4]);
        printf("   12 : numb_areas    = %d\n", numb_areas);
        printf("   13 : print results : ");
        if(print_on == 1) printf("yes\n"); else printf("no\n");
        printf("  ESC : Return to main MENU\n");

        printf("\n");
        show_arrow_bar();
        menu_select(&item);
        switch(item)        /* Submenu loop 2*/
        {
            case 1: printf("\n\nEnter new value: ");
                    displacement = get_par(0,512); break;
            case 2: printf("\n\nEnter new value: ");
                    diff_x_min0 = get_par(0,512); break;
            case 3: printf("\n\nEnter new value: ");
                    sum1_max = get_par(0,512); break;
            case 4: printf("\n\nEnter new value: ");
                    sum2_min = get_par(0,512); break;
            case 5: printf("\n\nEnter new value: ");
                    sum2_max = get_par(0,512); break;
            case 6: direction = -direction; break;
            case 7: printf("\n\nEnter new value: ");
                    stripe_x[0] = get_par(0,512); break;
            case 8: printf("\n\nEnter new value: ");
                    stripe_x[1] = get_par(0,512); break;
            case 9: printf("\n\nEnter new value: ");
                    stripe_x[2] = get_par(0,512); break;
            case 10: printf("\n\nEnter new value: ");
                     stripe_x[3] = get_par(0,512); break;
            case 11: printf("\n\nEnter new value: ");
                     stripe_x[4] = get_par(0,512); break;
            case 12: numb_areas = get_par(0,512);
                     for(area = 0; area < numb_areas; area++)
                     {
                         stripe_x[area] =
                             ((2*area+1)*PIXELS_PER_SCANLINE)/(2*numb_areas);
                     }
                     break;
            case 13: print_on = -print_on; break;
            case 0: return;
        }
    }
}



/*-main-program------------------------------------------------------*/

alg4()
/*-------------------------------------------------------------------*
   Mainprogram of the algorithm 'alg4'

   demo_switch = 0 for real time
               = 1 for single step
 *-------------------------------------------------------------------*/
{
    if(demo_switch == 1) adi_cpssel(VDB,VDB,VDB,VDB);
    else adi_cpssel(VDC,VDC,VDC,VDC);  /* to show the image of the camera */

    initialization();
```

```
    if(put_data_in_field())
    {
        main_program();
        if(demo_switch == 1)
        {
            printf("\nAreas shown in red [press return]");
            DrawAreas();
            getch();
            for(i=0; i<14; i++) putchar(8);
            printf("                \n");

            printf("Stripes shown in orange [press return]");
            for(area = 0; area < numb_areas; area++) DrawStripe(area);
            getch();
            for(i=0; i<14; i++) putchar(8);
            printf("                \n");

            printf("DOG filter response shown in green [press return]");
            for(area = 0; area < numb_areas; area++) DrawConvol(area);
            getch();
            for(i=0; i<14; i++) putchar(8);
            printf("                \n");

            printf("Seam points shown in blue [press return]");
            for(area = 0; area < numb_areas; area++) DrawResult(area);
            getch();
            for(i=0; i<14; i++) putchar(8);
            printf("                \n");
            fb_clf(B2,0);
        }
        else
        {
            for(area = 0; area < numb_areas; area++) DrawResult(area);
        }

        /* if(demo_switch == 1) adi_cpssel(VDC,VDC,VDC,VDC); */
    }
}



put_data_in_field()
/*-------------------------------------------------------------------------*
   function: read coordinates from file and put them in cse data-
             structure
   paramtrs: _filename: name of the file with the x- and y-coordinates
   document: none
   external
   routines: none
 *-------------------------------------------------------------------------*/
{
    fentry *fp;
    int cntr = 0;

    fldptr = fld;
    fp = featdata;
    while(cntr < nrfeat)  /* nrfeat = number of features: see fsmodl.c */
    {
        if(fldptr < fldptr_max)
        {
            (*fldptr).x = ((*fp).x) - XCOMP2;
            (*fldptr).y = ((*fp).y) - YCOMP2;
            if((*fldptr).y < NUMB_SCANLINES) fldptr++;  /* <--------- */
            fp++;
            cntr++;
        }
        else
        {
            printf("function __put_data_in_field__: too many features\n");
            return(0);
        }
    }
    return(1);
}



main_program()
/*-------------------------------------------------------------------------*
 *-------------------------------------------------------------------------*/
{
    initialization();
    phase = 2;

    while(phase < 5)
    {
        switch(phase)
        {
            case 2:
            {
                if(distr_ptr < fldptr)
                {
                    insert_new_pixel();
                    distr_ptr++;
                }
                else phase = 3;
                break;
            }
            case 3:
            {
                for(area = 0; area < numb_areas; area++)
                {
                    if((helpvar = length[area]) == 0) x = 0;
                    else x = stripe[area][helpvar-1];
                    for(y = length[area]; y < Y_MAX; y++) stripe[area][y] = x;
                    length[area] = Y_MAX;
                }
                phase = 4;
                break;
            }
            case 4:
            {
                h = 0;
                for(area = 0; area < numb_areas; area++)
                {
                    h = h + flag[area];
                }
                if(h == 0) phase = 5;
                break;
            }
        }
        filter();
    }

    /*-------------------------------------------------------------------------*
       Procedure to find step response complex of DOG-filter in signal
       state = 0    nothing found in the stripe-signal yet.
               1    low going pulse found, going to find maximum.
               2    maximum found, going to find minimum.
               3    step response complex found, now find (x,y) of ovelap
                    seam and store it in seam_elem data structure.
     *-------------------------------------------------------------------------*/
```

```
for(area = 0; area < numb_areas; area++)
{
    if(direction == -1)
    {
        state = 0;
        max = 0; min = 0;
        for(y = 16; y < Y_MAX; y++)
        {
            switch(state)
            {
                case 0:
                {
                    if((max = convol[area][y]) > diff_x_min0)
                    {
                        state = 1; break;
                    }
                }
                case 1:
                {
                    if((helpvar = convol[area][y]) >= max)
                    {
                        max = helpvar; break;
                    }
                    else
                    {
                        y_max = y; state = 2; min = max;
                        break;
                    }
                }
                case 2:
                {
                    if((helpvar = convol[area][y]) <= min)
                    {
                        min = helpvar;
                        break;
                    }
                    else
                    {
                        suml = max + min;
                        if(suml < 0) suml = -suml;
                        sum2 = max - min;

                        if(suml > suml_max)
                        {
                            state = 0; break;
                        }
                        if(sum2 < sum2_min)
                        {
                            state = 0; break;
                        }
                        if(sum2 > sum2_max)
                        {
                            state = 0; break;
                        }
                        /* state = 3 */
                        x = (stripe[area][y_max] + stripe[area][y_max-15]) >> 1;
                        for(i = y_max; i >= y_max-15; i--)
                        {
                            if(stripe[area][i] >= x)
                            {
                                helpvar = numb_seam_elem[area]++;
                                seam_elem[area][helpvar].y = i;
                                seam_elem[area][helpvar].x = x;
                                state = 0; break;
                            }
                        }
                        break;
                    }
                }
            }
        }
    }
}
else
{
    state = 0;
    max = 0; min = 0;
    for(y = 16; y < Y_MAX; y++)
    {
        switch(state)
        {
            case 0:
            {
                if((max = -convol[area][y]) > diff_x_min0)
                {
                    state = 1; break;
                }
            }
            case 1:
            {
                if((helpvar = -convol[area][y]) >= max)
                {
                    max = helpvar; break;
                }
                else
                {
                    y_max = y; state = 2; min = max;
                    break;
                }
            }
            case 2:
            {
                if((helpvar = -convol[area][y]) <= min)
                {
                    min = helpvar;
                    break;
                }
                else
                {
                    suml = max + min;
                    if(suml < 0) suml = -suml;
                    sum2 = max - min;

                    if(suml > suml_max)
                    {
                        state = 0; break;
                    }
                    if(sum2 < sum2_min)
                    {
                        state = 0; break;
                    }
                    if(sum2 > sum2_max)
                    {
                        state = 0; break;
                    }
                    x = (stripe[area][y_max] + stripe[area][y_max-15]) >> 1;
                    for(i = y_max; i >= y_max-15; i--)
                    {
                        if(stripe[area][i] <= x)
                        {
                            helpvar = numb_seam_elem[area]++;
                            seam_elem[area][helpvar].y = i;
```

```
                                seam_elem[area][helpvar].x = x;
                                state = 0; break;
                            }
                        }
                        break;
                    }
                }
            }
        }
    }

    /* coordinaten transformation */

    if(print_on == 1) printf("\n");
    for(area = 0; area < numb_areas; area++)
    {
        for(j = 0; j < numb_seam_elem[area]; j++)
        {
            seam_elem[area][j].z = origin_z +
                z_adjust[area]*(seam_elem[area][j].x - (float)area_bound[area]);
            seam_elem[area][j].x = origin_x + x_adjust*seam_elem[area][j].x;
            seam_elem[area][j].y = origin_y + y_adjust*seam_elem[area][j].y;

            if(print_on == 1)
                printf(" area %d:  (x,y,z) = (%5.3f,%5.3f,%5.3f)\n",
                area,seam_elem[area][j].x, seam_elem[area][j].y, seam_elem[area][j].z);
        }
        if(j == 0)
            if(print_on == 1)
                printf(" area %d.  no seam element found this area\n", area);
    }
}



filter()
/*-------------------------------------------------------------------------*
   Discrete DOG-filter
   Filter constants stored in dog_kernel[]
 *-------------------------------------------------------------------------*/
{
    for(area = 0; area < numb_areas; area++)
    {
        if(updated[area] < length[area])
        {
            if(length[area] >= 16)
            {
                y_coord = updated[area];
                sum = 0;
                for(k = 0; k < 16; k++)
                    sum = sum + stripe[area][y_coord+k-15]*dog_kernel[k];
                convol[area][y_coord] = (int)sum;
            }
            updated[area]++;
        }
        if(updated[area] == length[area]) flag[area] = 0;
        else flag[area] = 1;
    }
}



initialization()
/*-------------------------------------------------------------------------*
    this function is executed while vsync is active
 *-------------------------------------------------------------------------*/
{
    fldptr_max = fld + FIELD_SIZE;

    distr_ptr = fld;

    /* get position of the camera */
    origin_x = 0;
    origin_y = 0;
    origin_z = 0;

    for(area = 0; area < numb_areas; area++)
    {
       length[area] = 0;
       updated[area] = 0;
       flag[area] = 0;
/*
       min[area] = 0;
       max[area] = 0;
       y_max[area] = 0;
       state[area] = 0;
*/
       if(area != numb_areas-1)
           area_bound[area] = (stripe_x[area] + stripe_x[area+1])/2;
       else area_bound[area] = PIXELS_PER_SCANLINE;

       numb_seam_elem[area] = 0;
       /* stripe_x[area] = registers laden via FIFO */
    }
}



insert_new_pixel()
{
    x_coord = (*distr_ptr).x;
    y_coord = (*distr_ptr).y;

    area = 0;
    area_ptr = area_bound;
    while(1)
    {
       if(x_coord <= *area_ptr) break;
       area++;
       area_ptr++;
    }

    if(area < numb_areas)
    {
        x_coord = x_coord - stripe_x[area];
        if((l = length[area]) == 0)
        {
            for(y = 0; y <= y_coord; y++)
                stripe[area][y] = x_coord;
            x_prev[area] = x_coord;
            y_prev[area] = y_coord;
            counter[area] = 0;
            length[area] = y_coord;
        }
        else
        {
            if(y_coord != y_prev[area])
            {
                if((h = abs(x_coord - x_prev[area])) < 0) h=-h;
                if((h < displacement) || (counter[area] == 30))
                {
```

```
                    for(y = y_prev[area]; y < y_coord; y++)
                        stripe[area][y] = x_prev[area];
                    stripe[area][y_coord] = x_coord;
                    x_prev[area] = x_coord;
                    y_prev[area] = y_coord;
                    counter[area] = 0;
                    length[area] = y_coord;
                }
                else
                {
                    counter[area]++;
                }
            }
        }
    }
}




/*-------------------------------------------------------------------------*/


show_field(offset, color, threshold)
/*-----------------------------------------------------------------------*
 *-----------------------------------------------------------------------*/
int offset, color, threshold;
{
    int x, y;
    struct field *p;

    p = fld;
    while(p < fldptr)
    {
        x = (*p).x + offset;
        y = (*p).y;
        line(B2, 0, x, y, x, y, color);
        p++;
    }
}



DrawStripe(area)
int area;
{
    int x, y;

    for(y = 0; y < Y_MAX; y++)
    {
        x = stripe[area][y] + stripe_x[area];
        line(B2, 0, x,y,x,y, 2);
    }
}



DrawConvol(area)
int area;
{
    int x0, y0, x1, y1;

    y1 = 15; x1 = stripe_x[area] + (convol[area][y1] >> 2);
    for(y0 = 16; y0 < Y_MAX; y0++)
    {
        x0 = stripe_x[area] + (convol[area][y0] >> 2);
        line(B2, 0, x0,y0,x1,y1, 5);
        x1 = x0; y1 = y0;
    }
}



DrawResult(area)
int area;
{
    int x, y, j;

    for(j = 0; j < numb_seam_elem[area]; j++)
    {
        x = stripe_x[area] + seam_elem[area][j].x;
        y = seam_elem[area][j].y;
        /* draw arrow */
        line(B2, 0, x,y,x+40,y, 8);
        line(B2, 0, x+40,y,x+46,y-3, 8);
        line(B2, 0, x+40,y,x+46,y+3, 8);
        line(B2, 0, x+46,y-3,x+46,y+3, 8);
    }
}



DrawAreas()
{
    int x, i;

    for(i = 0; i < numb_areas; i++)
    {
        x = area_bound[i];
        line(B2, 0, x,0,x,256, 1);
    }
}
```

```
P1=\user\seam\work\fsmod
P2=\user\bin\fsmod
LIBS= cfg150ml.lib itx150ml.lib


$(P2)0.obj : $(P1)0.c
        cl /c /AL /Fo$(P2)0.obj $(P1)0.c

$(P2)1.obj : $(P1)1.c
        cl /c /AL /Fo$(P2)1.obj $(P1)1.c

$(P2)2.obj : $(P1)2.c
        cl /c /AL /Fo$(P2)2.obj $(P1)2.c

$(P2)3.obj : $(P1)3.c
        cl /c /AL /Fo$(P2)3.obj $(P1)3.c

fsdemo.exe:  $(P2)0.obj $(P2)1.obj $(P2)2.obj $(P2)3.obj
   link  $**,$@,,$(LIBS);
```

Appendix K: Listing image processing software: ADSP-2100 system

The contents of this appendix are:

adsp_fs.sys	system file:	contains hardware specification
header.dsp	header file:	contains initializations and the interrupt serviceroutines (written in the ADSP-2100 assembly code)
adsp_fs.c	program:	contains the image processing algorithms (written in the C programming language)
adsp_fs.hlp	help file:	contains the discriptions of the constants, variables and the functions defined in the source files: header.dsp and adsp_fs.c
m.bat	macro:	to compile the source files listed above in a more user friendly way

```
{-------------------------------------------------------------------------
   Source file name: adsp_fs.sys
   Author          : Werner de Deckere
   Date            : Januari 1990

   Eindhoven University of Technology
   Faculty of Electrotechnical Engineering
   Department Measurement and Control
   P.O. Box 513, 5600 MB Eindhoven, The Netherlands
 -------------------------------------------------------------------------

  The following on-board peripheral areas are selected:

   - 6k of RAM segments in the program memory section (PM)
   - 2K of RAM segments in the program memory section (PM)
   - 10K of RAM segments in the data memory section (DM)
   - 4 ports, memory mapped in the data memory section (DM),
     namely: TRIGGER,
             PHASEREG,
             XREG and
             YREG

  This file is a source file for the system builder program. The system
  builder needs the information in this file to create the architecture
  file.
  The following command has to be used to create the architecture file
  named adsp_fs.ach:

     dsppb adsp_fs.sys

 -------------------------------------------------------------------------}
.system                          adsp_fs;


{-------------------------------------------------------------------------
   Program Memory
 -------------------------------------------------------------------------}
.seg/ram/abs=h#0000/pm/code      PM_CODE[h#1800];  { program section }
.seg/ram/abs=h#4000/pm/data      PM_DATA[h#0800];  { data section }


{-------------------------------------------------------------------------
   Data Memory
 -------------------------------------------------------------------------}
.seg/ram/abs=h#0000/dm/data      DM_DATA[h#2800];
.port/abs=h#2800                 SCOPE_TRIGGER;
.port/abs=h#38FF                 PHASEREG;
.port/abs=h#3FFE                 XREG;
.port/abs=h#3FFF                 YREG;


{-------------------------------------------------------------------------
   Jumpers: ###
 -------------------------------------------------------------------------}
{
   data memory section   E1      ###         ram selected (2K)
                         E2      ###         ram selected (2K)
                         E3      ###         ram selected (2K)
                         E4      ###         ram selected (2K)
                         E5      ###         ram selected (2K)
                         E6      . ###       scope trigger
                         E7      . . .       prototyping connector
                         E8      . . .       prototyping connector
                         E9      . .         interrupt switch irq2 disabled
                         E10     . .         cyclic interrupt irq3 disabled
}

.endsys;
```

```
{-------------------------------------------------------------------------
   Source file name: header.dsp
   Author          : Werner de Deckere
   Date            : Januari 1990

   Eindhoven University of Technology
   Faculty of Electrotechnical Engineering
   Department Measurement and Control
   P.O. Box 513, 5600 MB Eindhoven, The Netherlands
 -------------------------------------------------------------------------

   This file contains the interrupt service routines for the image
   processing software described in the source file: adsp_fs.c
   The interrupt service routine are written in the ADSP-2100 assembly
   code. To generate a object file use the ADSP-2100 assembler:

     dsppa -c header.dsp

   Link the generated object file with the object file of the source
   file: adsp_fs.c

     link header adsp_fs -a adsp_fs -c -e adsp_fs -g -x

 -------------------------------------------------------------------------}



.module/abs=0/ram   RTH;

{-------------------------------------------------------------------------
   Definition of the ports (refer to source file: adsp_fs.sys)
 -------------------------------------------------------------------------}
.port XREG;                          { Register on coordinate generator: XREG }
.port YREG;                          { Register on coordinate generator: YREG }
.port SCOPE_TRIGGER;                 { Register on ADSP-2100 evaluation board }


{-------------------------------------------------------------------------
   Definition of the variables used in the interrupt service routines
 -------------------------------------------------------------------------}
.var/ram/dm/abs=0 phase_;            { Refer to source file: adsp_fs.c        }
.var/ram/dm/abs=1 errorcode_;        { Refer to source file: adsp_fs.c        }
.var/ram/dm count_, count_max_;      { Refer to source file: adsp_fs.c        }

.var/ram/dm store0_;                 { Reservation for temporary storage      }
.var/ram/dm store1_;                 { Reservation for temporary storage      }
.var/ram/dm store2_;                 { Reservation for temporary storage      }
.var/ram/dm store3_;                 { Reservation for temporary storage      }
.var/ram/dm store4_;                 { Reservation for temporary storage      }


{-------------------------------------------------------------------------
   Make variables global, so that they can be accessed by routines in
   routines in other source files
 -------------------------------------------------------------------------}
.global SCOPE_TRIGGER;
.global phase_, errorcode_;
.global count_, count_max_;
.global store0_, store1_, store2_, store3_, store4_;


{-------------------------------------------------------------------------
   Define variables, that already have been defined in other source files
 -------------------------------------------------------------------------}
.external pixptr_, pixptr_max_;      { Refer to source file: adsp_fs          }
.external main_;                     { Refer to source file: adsp_fs          }
.external ____top_of_ram;            { The variable ____top_of_ram is defined }
                                     { by the C-compiler. For more details    }
                                     { refer to ADSP-2100 Cross-software      }
                                     { manual.                                }


{-------------------------------------------------------------------------
   Program
 -------------------------------------------------------------------------}

vector_interrupt_table:              { Address 0 in the ADSP-2100 data memory }
          jump serv_rout0;           { Interrupt input int0: dogint           }
          jump serv_rout1;           { Interrupt input int1: vsync            }
          jump serv_rout2;           { Interrupt input int2: start            }
          rti;                       { Interrupt input int3 is not used       }
start_address:                       { Address 4 in the ADSP-2100 data memory }
          IMASK = b#0000;            { IMASK = interrupt mask register        }
                                     { bit0-3 = 0: disable all interrupts     }
          ICNTL = b#01001;           { ICNTL = interrupt control register     }
                                     { bit0 = 1: edge sensitive interrupt     }
                                     { bit4 = 0: level sensitive interrupt    }
                                     { bit4 = 0: level sensitive interrupt    }
                                     { bit4 = 1: edge sensitive interrupt     }
                                     { bit4 = 0: no interrupt nesting         }
          MSTAT = b#0000;            { MSTAT = mode status register           }
                                     { bit0 = 0: primary data register bank   }
                                     { bit1 = 0: no bit reverse mode (DAG1)   }
                                     { bit2 = 0: no ALU overflow latch mode   }
                                     { bit3 = 0: no AR saturation mode        }

          m4 =  ^____top_of_ram;     { Frame pointer of the software stack    }
          i4 =  ^____top_of_ram-1;   { Stack pointer of the software stack    }
          l0 =  0;                   { Initialisation of length registers     }
          l1 =  0;
          l2 =  0;
          l3 =  0;
          l4 =  0;
          l5 =  0;
          l6 =  0;
          l7 =  0;
          m0 =  0;                   { initialisation of the modify registers }
          m1 =  1;
          m2 =  0;
          m3 = -1;
          m5 =  1;
          m7 = -1;

          call main_;                { Call function main() in the source file }
                                     { adsp_fs().                             }
          trap;                      { Stop execution program on the ADSP-2100.}
          jump start_address;


{-------------------------------------------------------------------------}
serv_rout0: ! dogint
          MSTAT = b#0001;            { bit0 = 1: secondary data register bank }
                                     { bit1 = 0: no bit reverse mode (DAG1)   }
                                     { bit2 = 0: no ALU overflow latch mode   }
                                     { bit3 = 0: no AR saturation mode        }
          ax0 = dm(pixptr_);         { Compare pixptr to pixptr_max.          }
          ay0 = dm(pixptr_max_);
          af =  ax0-ay0;
          if ge jump exceeded;       { If pixptr is larger than pixptr_max then }
                                     { go to label 'exceeded'.                }
          ax0 = dm(XREG);            { Store contents of register XREG in ax0.  }
          ay0 = dm(YREG);            { Store contents of register YREG in ay0.  }
```

```
                dm(store0_) = i0;       { Save contents of i0 and m0.           }
                dm(store1_) = m0;

                i0 = dm(pixptr_);       { The address, where pixptr points at, is  }
                                        { loaded in the index register i0.         }
                m0 = 1;                 { Modify register m0 is equal to 1.        }
                dm(i0,m0) = ax0;        { The contents of ax0 are stored at the    }
                                        { address where i0 points at. The address  }
                                        { stored in i0 is increased automatically  }
                                        { with the contents of m0 (=1).            }
                dm(i0,m0) = ay0;        { Idem for the contents of register ay0.   }
                dm(pixptr_) = i0;       { Save increased address in pixptr.        }

                i0 = dm(store0_);       { Restore i0 and m0.
                m0 = dm(store1_);
                rti;                    { Return from interrupt service routine.   }

exceeded:       si = 1;                 { There were to much interrupt requests.   }
                dm(errorcode_) = si;    { Set errorcode to 1.                      }
                rti;                    { Return from interrupt service routine.   }


{-------------------------------------------------------------------------------}
serv_rout1: ! vsync
                MSTAT = b#0001;         { bit0 = 1: secondary data register bank   }
                                        { bit1 = 0: no bit reverse mode (DAG1)     }
                                        { bit2 = 0: no ALU overflow latch mode     }
                                        { bit3 = 0: no AR saturation mode          }
                ax0 = dm(phase_);
                ay0 = 1;
                af = ax0-ay0;
                if eq jump phase1;      { If (phase==1) then jump to label phase1. }
                ay0 = 3;
                af = ax0-ay0;
                if eq jump phase3;      { If (phase==3) then jump to label phase3. }
                ay0 = 2;
                af = ax0-ay0;
                if eq rti;              { If (phase==2) then perform rti.          }
                ay0 = 4;
                af = ax0-ay0;
                if eq rti;              { If (phase==4) then perform rti.          }

                si = 2;                 { Undefined situation: set errorcode to 2. }
                dm(errorcode_) = si;
                rti;                    { Return from interrupt service routine.   }

phase1:         ax0 = dm(count_);
                ay0 = dm(count_max_);
                af  = ax0-ay0;
                if lt jump incr_cnt1;   { If count is less than count_max then     }
                                        { increase count: label incr_cnt1.         }
                si = 2;                 { Interrupt accepted: set phase to 2.      }
                dm(phase_) = si;
                rti;                    { Return from interrupt service routine    }

phase3:         ax0 = dm(count_);
                ay0 = dm(count_max_);
                af  = ax0-ay0;
                if lt jump incr_cnt1;   { If count is less than count_max then     }
                                        { increase count.                          }
                si = 4;                 { Interrupt accepted: set phase to 4.      }
                dm(phase_) = si;
                rti;

incr_cnt1:      ay0 = dm(count_);       { Increase contents of count with 1.       }
                ar = ay0+1;
                dm(count_) = ar;
                rti;                    { Return from interrupt service routine.   }


{-------------------------------------------------------------------------------}
serv_rout2: ! start
                MSTAT = b#0001;         { bit0 = 1: secondary data register bank   }
                                        { bit1 = 0: no bit reverse mode (DAG1)     }
                                        { bit2 = 0: no ALU overflow latch mode     }
                                        { bit3 = 0: no AR saturation mode          }
                ax0 = dm(count_);
                ay0 = dm(count_max_);
                af  = ax0-ay0;
                if lt jump incr_cnt2;   { If count is less than count_max then     }
                                        { increase count: label incr_cnt1.         }
                si = 1;                 { Interrupt accepted: set phase to 4.      }
                dm(phase_) = si;
                rti;                    { Return from interruptservice routine.    }

incr_cnt2:      ay0 = dm(count_);       { Increase contents of count with 1        }
                ar = ay0+1;
                dm(count_) = ar;
                rti;                    { Return from interruptservice routine.    }


{-------------------------------------------------------------------------------}
!
!
!   Here below you'll find the interrupt service routines translated to
!   the C programming language. For the assembly instructions, that could
!   not be translated to a valid C instruction, a pseudo instruction is
!   used:  PseudoInstruction( ... ).
!
!
!     service_routine_0()  <--- interrupt int0: dogint
!     {
!        PseudoInstruction( Switch to second register bank );
!         if(pixptr < pixptr_max)
!         {
!             PseudoInstruction( Save registers i0 and m0 );
!             (*pixptr).x =
!                 PseudoInstruction( Get contents of XREG register );
!             (*pixptr++).y =
!                 PseudoInstruction( Get contents of YREG register );
!             PseudoInstruction( Restore registers i0 and m0 );
!         }
!         else errorcode = 1;
!         PseudoInstruction( Return from interupt service routine );
!     }
!
!
!     service_routine_1()  <--- interrupt int1: vsync
!     {
!         PseudoInstruction( Switch to second register bank );
!         switch(phase)
!         {
!             case 1: if(count < count_max) count++;
!                     else phase = 2; break;
!             case 2: break;
!             case 3: if(count < count_max) count++;
!                     else phase = 4; break;
!             case 4: break;
```

```
!             default: errorcode = 2; break;
!         }
!         PseudoInstruction( Return from interupt service routine );
!     }
!
!
!
!     service_routine_2()  <--- interrupt int2: start
!     {
!         PseudoInstruction( Switch to second register bank );
!         if(count < count_max) count++;
!         else phase = 1;
!         PseudoInstruction( Return from interupt service routine );
!     }
!
!
!
.endmod;
```

```
/*-------------------------------------------------------------------------*

   Source file name: adsp_fs.c    (algorithm for finding overlap seams)
   Author          : Werner de Deckere
   Date            : Januari 1990

   Eindhoven University of Technology
   Faculty of Electrotechnical Engineering
   Department Measurement and Control
   P.O. Box 513, 5600 MB Eindhoven, The Netherlands

 *-------------------------------------------------------------------------*

   This source file contains the algorithms to find an overlap seam in a
   piece of work. The C programming langueage is used to describe the
   algorithms.
   To compile the sourcefile, use the ADSP-2100 C Compiler:

     cc2100 adsp_fs -m

   The object file, generated by the C-compiler should be linked with the
   object file of the source file header.dsp:

     link header adsp_fs -a adsp_fs -c -e adsp_fs -g -x

 *-------------------------------------------------------------------------*/



/*-------------------------------------------------------------------*
   Definition of constants

    TABLE_SIZE                    length of the pixel table array
    PIXELS_PER_SCANLINE            number of pixels on a scanline
    MAX_NUMB_AREAS           maximum number of areas in the screen
    X_MIN     minimum x coordinate, refer to distribution_module()
    X_MAX     maximum x coordinate, refer to distribution_module()
    Y_MAX                              maximum y coordinate
 *-------------------------------------------------------------------*/
#define TABLE_SIZE                          2500
#define PIXELS_PER_SCANLINE                  604
#define MAX_NUMB_AREAS                         5
#define X_MIN                                 50
#define X_MAX                                550
#define Y_MAX                                350



/*-----------------------------------------------------------*
   variables for the distribution module:
   ( menu variables: ) displacement, smooth_factor.

   Remark: In this program it is preferable to use global
           variables in stead of local variables.
           The reason for this is that global varibles can be
           accessed much more faster that local variables.
           The access time of global variables is always two
           instruction cycles while the access time of local
           variables is always larger than two cycles. Note
           that local variables are stored on the software
           stack, starting at address ____top_of_main.
           (defined in source file: header.dsp )
 *-----------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*
   Pixel table data structure
 *-------------------------------------------------------------------*/
struct field
{
    int x,                /* x- and y-coordinate                     */
        y;
};
struct field pixel_table[TABLE_SIZE];
struct field *pixptr,     /* Pointer, points to empty place in the */
                          /* pixel table.                          */
             *pixptr_max, /* Pointer, points to the end of the     */
                          /* pixel table.                          */
             *distr_ptr;  /* Pointer, points to the element of the */
                          /* pixel table, that is to be processed. */
int numb_pixels;          /* Number of pixels in the pixel table.  */


/*-----------------------------------------------------------*
   stripe and filtered stripe (convol) data structures.
 *-----------------------------------------------------------*/
int stripe[MAX_NUMB_AREAS][Y_MAX];  /* stripe                      */
int convol[MAX_NUMB_AREAS][Y_MAX];  /* filtered stripes            */
int *ptr, *help_ptr,          /* pointers to the stripe and convol */
    *stripe_ptr,              /* data structures                   */
    *convol_ptr;
int length[MAX_NUMB_AREAS],   /* the stripe's length               */
    x_prev[MAX_NUMB_AREAS],   /* the stripe's previous x coordinate */
    y_prev[MAX_NUMB_AREAS],   /* the stripe's previous y coordinate */
    counter[MAX_NUMB_AREAS]; /* refer to distibution_module()      */


/*-----------------------------------------------------------*
   seam element data structure.
 *-----------------------------------------------------------*/
struct seam_elem_record
{
    float x,      /* x-, y- and z-coordinate */
          y,
          z;
};
struct seam_elem_record seam_elem[MAX_NUMB_AREAS][16],
                        *seam_elem_ptr;
int numb_seam_elem[MAX_NUMB_AREAS]; /* number of seam elements */



/*-----------------------------------------------------------*
   process parameters
 *-----------------------------------------------------------*/
int displacement;   /* refer to function: insert_new_pixel() */
int smooth_factor;  /* idem                                  */
int stripe_type;    /* idem,  1: stripe type a               */
                    /*       -1: stripe type b               */

int threshold;      /* refer to function: detection_module() */
long diff_max;      /* idem                                  */
long sum_min;       /* idem                                  */
long sum_max;       /* idem                                  */

int direction;      /* direction: equal to -1 or 1           */
int numb_areas;     /* numb_areas <= MAX_NUMB_AREAS          */
```

```
/*-------------------------------------------------------------*
   variables for area definition:
 *-------------------------------------------------------------*
xxx numb_areas                  (process parameter)             */
int stripe_x[MAX_NUMB_AREAS];   /* Estimated position (x coordinate) */
                                /* of the stripes of light         */
/* refer also to stripe_z (coordinate transformation module)      */

int area_bound[MAX_NUMB_AREAS]; /* Boundaries of the area. These   */
                                /* are calculated in the function  */
                                /* initialization()                */
int area,
    *area_ptr,
    bound;                      /* boundary                        */


/*-------------------------------------------------------------*
   variables for the distribution module:
 *-------------------------------------------------------------*
xxx displacement      (process parameter)
xxx smooth_factor     (process parameter)                    */


/*-------------------------------------------------------------*
   variables for the filter module:
 *-------------------------------------------------------------*/
int pm dog_kernel[16] = { -1, -3, -3, -1,  0,  1,  3,  4,
                          4,  3,  1,  0, -1, -3, -3, -1};
   /* Filter coefficients of the discrete DOG filter.       */
   /* Note that the array dog_kernel is located in the data */
   /* section of the program memory.                        */


/*-------------------------------------------------------------*
   variables for detection module:
 *-------------------------------------------------------------*
xxx threshold         (process parameter)
xxx diff_max          (process parameter)
xxx sum_min           (process parameter)
xxx sum_max           (process parameter)                */
int state;          /* See description in the function   */
int min, max;       /* minimum and maximum               */
int sum, diff;      /* sum and difference                */
int y_max;          /* y coordinate where maximum occurs */


/*-------------------------------------------------------------*
   variables for coordinate transformation module:
 *-------------------------------------------------------------*/
float origin_x,             /* camera's position (x-, y- and z   */
      origin_y,             /* coordinate)                       */
      origin_z;
float labda,                /* elements of the scaling matrix F  */
      mu,
      nu;
float stripe_z[MAX_NUMB_AREAS],
      z_adjust[MAX_NUMB_AREAS];
/* stripe_z = z-coordinate of the reference stripe (stripe_x) */
/* z_adjust = Dependent on the positioning of the projector   */
/*            with regard to the camera; z_adjust is equal to */
/*            1 devided by tan(alpha). Alpha is the angle     */
/*            between the axis of the pjector and the axis of */
/*            the camera.                                     */
```

```
/*-------------------------------------------------------------*
   some other global variables.
 *-------------------------------------------------------------*/
int x_coord, y_coord;
int l, h;
int i, j;
int x, y;
int x_previous, y_previous;
int helpvar, k;
int t;



/*-------------------------------------------------------------*
   global variables that were defined externally, i.e. variables
   that were defined in the source file: header.dsp
 *-------------------------------------------------------------*/
extern int phase;
    /*-------------------------------------------------------------*
       program phase:

       phase == 0: Start phase; wait for start pulse on input int2.
                1: Start pulse detected; wait for first vsync pulse
                   on input int1.
                2: First vsync pulse detected; initialise variables.
                3: Wait for second vsync pulse on input int1. If the
                   service routine 0 is not executed, the pixels, that
                   are stored in the pixel table untill then, are
                   distributed over the areas.
                4: Second vsync pulse detected. Finish the distribution
                   of the pixels stored in the pixel table.
                5: Perform function add_tails().
                6: Perform filter operation: filter_module()
                7: Perform function detection_module().
                8: Perform coordinate transformation:
                      coordinate_transf_module()
     *-------------------------------------------------------------*/
extern int errorcode;
    /*-------------------------------------------------------------*
       If an error occurs the corresponding error number is stored
       in the variable errorcode:

       errorcode == 0: No errors.
                    1: To many pixels in the videosignal. Pixel table
                       to small to store all pixels.
                    2: Undefined phase detected in interrupt service
                       routine number 1.
     *-------------------------------------------------------------*/
extern int count,       /* Number of interrupts detected.                */
           count_max;   /* Level interrupt is accepted when the contents */
                        /* of the variable count exceeds count_max.      */
extern int store0,      /* variables for temporary storage.              */
           store1,
           store2,
           store3,
           store4;
/*
#pragma ADSP2100
.external SCOPE_TRIGGER;
#pragma ADSP2100
*/
```

```
/*-------------------------------------------------------------------------*
  Functions
 *-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*
   The main program can be be interrupted by interrupt requests on the
   inputs int0, int1 or int2. In that case the corresponding interrupt
   service routine will be called. These routines are written in the
   ADSP-2100 assembly language and these are listed in the source file:
   header.dsp
   Here below you'll find the interrupt service routines translated to
   the C programming language. For the assembly instructions, that could
   not be translated to a valid C instruction, a pseudo instruction is
   used:  PseudoInstruction( ... ).
 *-------------------------------------------------------------------------*
    service_routine_0()  <--- interrupt int0: doint
    {
        PseudoInstruction( Switch to second register bank );
        if(pixptr < pixptr_max)
        {
            PseudoInstruction( Save registers i0 and m0 );
            (*pixptr).x =
                PseudoInstruction( Get contents of XREG register );
            (*pixptr++).y =
                PseudoInstruction( Get contents of YREG register );
            PseudoInstruction( Restore registers i0 and m0 );
        }
        else errorcode = 1;
        PseudoInstruction( Return from interupt service routine );
    }


    service_routine_1()  <--- interrupt int1: vsync
    {
        PseudoInstruction( Switch to second register bank );
        switch(phase)
        {
            case 1: if(count < count_max) count++;
                    else phase = 2; break;
            case 2: break;
            case 3: if(count < count_max) count++;
                    else phase = 4; break;
            case 4: break;
            default: errorcode = 2; break;
        }
        PseudoInstruction( Return from interupt service routine );
    }


    service_routine_2()  <--- interrupt int2: start
    {
        PseudoInstruction( Switch to second register bank );
        if(count < count_max) count++;
        else phase = 1;
        PseudoInstruction( Return from interupt service routine );
    }
 *-------------------------------------------------------------------------*/



initialization()
/*-------------------------------------------------------------------------*
   function         initialization()
   version          1.0
   date             nov 1989
   author           Werner de Deckere
   institute        Eindhoven University of Technology
                    Department of Electrical Engineering
   group            Measurement and Control
   address          P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

   parameters (+ = entry parameter, - = exit parameter)
      none

   return code
      none

   called subroutines library
      empty

   subroutine description
      In this function a number of variables is initialised

   declarations
      none

 *-------------------------------------------------------------------------*/
{
    pixptr_max    = pixel_table + TABLE_SIZE;
    pixptr        = pixel_table;
    distr_ptr     = pixel_table;
    numb_pixels   = 0;


    /* load values following variables via FIFO register: */
    numb_areas    = 3;
    stripe_x[0]   = 180;
    stripe_x[1]   = 285;
    stripe_x[2]   = 400;
    stripe_x[3]   = 450;
    stripe_x[4]   = 500;

    stripe_z[0]   = 10;
    stripe_z[1]   = 10;
    stripe_z[2]   = 10;
    stripe_z[3]   = 10;
    stripe_z[4]   = 10;

    origin_x      = 0;
    origin_y      = 0;
    origin_z      = 0;

    labda         = 1;
    mu            = 1;
    nu            = 1;

    z_adjust[0]   = 10;
    z_adjust[1]   = 10;
    z_adjust[2]   = 10;
    z_adjust[3]   = 10;
    z_adjust[4]   = 10;

    displacement  = 40;
    smooth_factor = 30;

    direction     = 1;
    stripe_type   = 1;
    threshold     = 12;
    diff_max      = 20;
```

```
    sum_min       = 40;
    sum_max       = 300;


    for(area = 0; area < numb_areas; area++)
    {
        x_prev[area] = 0;
        y_prev[area] = -1;
        counter[area] = 0;

        /* calculate area boundaries */
        if(area != numb_areas-1)
            area_bound[area] = (stripe_x[area] + stripe_x[area+1]) >> 1;
            /* Division by 2 is performed by a shift operation of one */
            /* bit in the direction of the least significant bit.     */
        else area_bound[area] = PIXELS_PER_SCANLINE;

        numb_seam_elem[area] = 0;
    }
}



main()
/*-------------------------------------------------------------------------*
   function         main()
   version          1.0
   date             nov 1989
   author           Werner de Deckere
   institute        Eindhoven University of Technology
                    Department of Electrical Engineering
   group            Measurement and Control
   address          P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

   parameters (+ = entry parameter, - = exit parameter)
      none

   return code
      none

   called subroutines library
      initialization(), insert_new_pixel(), filter_module(),
      detection_module(), coord_transf_module()

   subroutine description
      This function is the main function of this seam finding algorithm.
      First in each area a mathematical function is generated using the
      elements in the pixel table. The function definitions are stored
      in the stripe data structure.
      Second the stripes can be convoluated with the DOG-filter kernel.
      Third the pattern recognition is done.
      And fourth the resulting coordinates are transformated from the
      sensor coordinate system to the world coordinate system.

   declarations
      none

 *-------------------------------------------------------------------------*/
{
        for(y=0;y<5;y++)
            for(x=0;x<350;x++) stripe[y][x] = y;

        while(1) /* infinite loop */
        {
            phase     = 0;
            errorcode = 0;
            count     = 0;
            count_max = 25;
#pragma ADSP2100
            IMASK=b#0100;          ( enable interrupt int2 (start) )
#pragma ADSP2100

            while(phase == 0);     /* wait for start pulse */
            if(errorcode != 0) break;
            /* phase 1 */

            /*-----------------------------------------------------------*
               the minimal width of the vsync pulse is
               5 half scanlines of 64 microsec each
                 = 64*5/2 microsec
                 = 40 microsec
                 = 40000 nanosec
                 = 500 instructions (a 80 nanosec)
                 = 320 instructions (a 125 nanosec)
               the number of instructions executed per interrupt
               service is equal to 15
             *-----------------------------------------------------------*/
            count     = 0;
            count_max = 10;
#pragma ADSP2100
            IMASK=b#0010;          ( enable interrupt int1 (vsync) )
#pragma ADSP2100

            while(phase == 1);     /* wait for the first vsync pulse */
#pragma ADSP2100
            dm(SCOPE_TRIGGER) = si;
            IMASK = b#0000;        ( disable all interrupts )
#pragma ADSP2100
            if(errorcode != 0) break;
            /* phase 2 */

            initialization();
            /*-----------------------------------------------------------*
               the maximal width of the vsync pulse is
               15 half scanlines of 64 microsec each
                 = 15*(64/2) microsec
                 = 120 microsec
                 = 120000 nanosec
                 = 1500 instructions (a 80 nanosec)
                 = 960  instructions (a 125 nanosec)
             *-----------------------------------------------------------*/
            for(t=0; t<400; t++);

            phase     = 3;
            count     = 0;
            count_max = 10;
#pragma ADSP2100
            IMASK=b#0011;          ( enable interrupts int0 and int1 )
#pragma ADSP2100

            while(phase == 3)      /* wait for the second vsync pulse */
            {
                if(distr_ptr < pixptr)
                {
                    insert_new_pixel();
                    distr_ptr++;
                }
            }

#pragma ADSP2100
            IMASK = b#0000;        ( disable all interrupts )
```

```
#pragma ADSP2100
            if(errorcode != 0) break;
            send_trigger_scope_pulse();
            /* phase 4: finish distribution operation */
            while(distr_ptr < pixptr)
            {
                insert_new_pixel();
                distr_ptr++;
            }

            phase = 5;
            send_trigger_scope_pulse();
            add_tails();

            phase = 6;
            send_trigger_scope_pulse();
            filter_module();

            phase = 7;
            send_trigger_scope_pulse();
            detection_module();

            phase = 8;
            send_trigger_scope_pulse();
            coordinate_transf_module();

            phase = 9;
            send_trigger_scope_pulse();

#pragma ADSP2100
            trap; { remove this instruction when running the }
                  { program continueously                    }
#pragma ADSP2100

        }
}



insert_new_pixel()
/*-------------------------------------------------------------------------*
   function         insert_new_pixel()
   version          1.0
   date             nov 1989
   author           Werner de Deckere
   institute        Eindhoven University of Technology
                    Department of Electrical Engineering
   group            Measurement and Control
   address          P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

   parameters (+ = entry parameter, - = exit parameter)
      none

   return code
      none

   called subroutines library
      empty

   subroutine description
      In this function one element of the pixel table is considered.
      First the area number is determinated. Second the element is
      added to the stripe of the determinated area.
      If the difference between the y_coordinate of the element and the
      y_coordinate its predecessor in the stripe data structure is larger
      than the variable displacement then the element is not added to the
      stripe data structure. Only when more than a certain number of these
      elements (specified in the variable smooth_factor) are found then
      the element is added to the stripe data structure. The effect is
      that sudden large changes in the description of the stripe are
      removed.

   declarations
      none

 *-------------------------------------------------------------------------*/
{
    if(numb_pixels++ == 0) return; /* ignore first pixel */
    x_coord = (*distr_ptr).x;
    y_coord = (*distr_ptr).y;

    if(x_coord <= X_MIN) return;
    if(x_coord >= X_MAX) return;

    /* determination of the area number */
    area = 0;
    area_ptr = area_bound;
    while(1)
    {
       if(x_coord <= *area_ptr++) break;
       area++;
    }

    /* smooth filtering */
    x_coord = x_coord - stripe_x[area];
    if(y_prev[area] == -1)
    {
        stripe_ptr = &stripe[area][0];
        for(y = 0; y <= y_coord; y++) *stripe_ptr++ = x_coord;

        x_prev[area] = x_coord;
        y_prev[area] = y_coord;
        counter[area] = 0;
    }
    else
    {
        if(y_coord > y_prev[area])
        {
            if(stripe_type == 1) x_previous = x_prev[area];
            else x_previous == x_coord;

            if((h = x_coord - x_previous) < 0) h=-h;
                /* h = abs(x_coord - x_previous) */
            if((h < displacement) || (counter[area] == smooth_factor))
            {
                y_previous = y_prev[area];
                stripe_ptr = &stripe[area][y_previous];
                for(y = y_previous; y < y_coord; y++)
                    *stripe_ptr++ = x_previous;

                stripe[area][y_coord] = x_coord;
                x_prev[area] = x_coord;
                y_prev[area] = y_coord;
                counter[area] = 0;
            }
            else
            {
                counter[area]++;
            }
        }
    }
```

```
}



add_tails()
/*-------------------------------------------------------------------------*
   function         add_tails()
   version          1.0
   date             nov 1989
   author           Werner de Deckere
   institute        Eindhoven University of Technology
                    Department of Electrical Engineering
   group            Measurement and Control
   address          P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

   parameters (+ = entry parameter, - = exit parameter)
      none

   return code
      none

   called subroutines library
      empty

   subroutine description
      After the distribution is done, the stripes are undefined in the part
      from the last pixel till the end. The function add_tails fills this
      part with the value equals to the x coordinate of the last pixel in
      the stripe array.

   declarations
      none

 *-------------------------------------------------------------------------*/
{
    for(area=0; area < numb_areas; area++)
    {
        if((helpvar = y_prev[area]) == -1) x = 0;
        else x = stripe[area][helpvar];
        stripe_ptr = &stripe[area][helpvar+1];
        for(y = helpvar+1; y < Y_MAX; y++) *(stripe_ptr++) = x;
    }
}



filter_module()
/*-------------------------------------------------------------------------*
   function         filter_module()
   version          1.0
   date             nov 1989
   author           Werner de Deckere
   institute        Eindhoven University of Technology
                    Department of Electrical Engineering
   group            Measurement and Control
   address          P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

   parameters (+ = entry parameter, - = exit parameter)
      none

   return code
      none

   called subroutines library
      empty

   subroutine description
      One element (selected by variables area) of each stripe element
      data structure is convuated with the DOG filter kernel.
      The filter constants are stored in the dog_kernel[] array.

   declarations
      none

 *-------------------------------------------------------------------------*/
{
    helpvar = Y_MAX - 16;
    for(area = 0; area < numb_areas; area++)
    {
        stripe_ptr = &stripe[area][0];
        convol_ptr = &convol[area][15];
#pragma ADSP2100
        dm(store2_) = i1;
        dm(store3_) = i5;
        dm(store4_) = mr0;
        cntr = dm(helpvar_);
        do conv_0 until ce;
            i1 = dm(stripe_ptr_);
            i5 = ^dog_kernel_;
            mr = 0, mx0 = dm(i1,m1), my0 = pm(i5,m5);
            dm(stripe_ptr_) = i1;
            cntr = 15;
            do conv_1 until ce;
conv_1:         mr = mr + mx0*my0(SS), mx0 = dm(i1,m1), my0 = pm(i5,m5);
            mr = mr + mx0*my0(SS);
            sr = lshift mr0 by -1 (lo);
            sr = sr or ashift mr1 by -1 (hi);
            i1 = dm(convol_ptr_);
            dm(i1,m1) = mr0;
conv_0:     dm(convol_ptr_) = i1;
        i1 = dm(store2_);
        i5 = dm(store3_);
        mr0 = dm(store4_);
#pragma ADSP2100
    }
}



detection_module()
/*-------------------------------------------------------------------------*
   function         detection_module()
   version          1.0
   date             nov 1989
   author           Werner de Deckere
   institute        Eindhoven University of Technology
                    Department of Electrical Engineering
   group            Measurement and Control
   address          P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

   parameters (+ = entry parameter, - = exit parameter)
      none

   return code
      none

   called subroutines library
      empty
```

```
   subroutine description
      This function tries to find the response in the convol data
      structure that is similar to the DOG-filter step response.
      The parameter direction determines whether the function has to
      look for positive or negative steps (positive or negative overlap
      seams).
      After a place is found where the response is similar to the
      DOG-filter step response, the function is going to look for
      the position of the overlap seam.

   declarations
      none

 *--------------------------------------------------------------------------*/
{
    /*--------------------------------------------------------------------*
       state = 0    nothing found in the stripe-signal yet.
               1    low going pulse found, going to find maximum.
               2    maximum found, going to find minimum.
               3    step response complex found, now find (x,y) of ovelap
                    seam and store it in seam_elem data structure.
     *--------------------------------------------------------------------*/

    for(area = 0; area < numb_areas; area++)
    {
        if(direction == -1)
        {
            state = 0;
            max = 0; min = 0;
            for(y = 16; y < Y_MAX; y++)
            {
                switch(state)
                {
                    case 0:
                    {
                        if((max = convol[area][y]) > threshold)
                        {
                            state = 1; break;
                        }
                    }
                    case 1:
                    {
                        if((helpvar = convol[area][y]) >= max)
                        {
                            max = helpvar; break;
                        }
                        else
                        {
                            y_max = y; state = 2; min = max;
                            break;
                        }
                    }
                    case 2:
                    {
                        if((helpvar = convol[area][y]) <= min)
                        {
                            min = helpvar;
                            break;
                        }
                        else
                        {
                            diff = max + min;
                            if(diff < 0) diff = -diff;
                            sum = max - min;

                            if(diff > diff_max)
                            {
                                state = 0; break;
                            }
                            if(sum < sum_min)
                            {
                                state = 0; break;
                            }
                            if(sum > sum_max)
                            {
                                state = 0; break;
                            }
                            /* state = 3 */
                            x = (stripe[area][y_max] + stripe[area][y_max-15]) >> 1;
                            for(i = y_max; i >= y_max-15; i--)
                            {
                                if(stripe[area][i] >= x)
                                {
                                    helpvar = numb_seam_elem[area]++;
                                    seam_elem[area][helpvar].y = (float)i;
                                    seam_elem[area][helpvar].x = (float)x;
                                    state = 0; break;
                                }
                            }
                            break;
                        }
                    }
                }
            }
        }
    }
}
else
{
    state = 0;
    max = 0; min = 0;
    for(y = 16; y < Y_MAX; y++)
    {
        switch(state)
        {
            case 0:
            {
                if((max = -convol[area][y]) > threshold)
                {
                    state = 1; break;
                }
            }
            case 1:
            {
                if((helpvar = -convol[area][y]) >= max)
                {
                    max = helpvar; break;
                }
                else
                {
                    y_max = y; state = 2; min = max;
                    break;
                }
            }
            case 2:
            {
                if((helpvar = -convol[area][y]) <= min)
                {
                    min = helpvar;
                    break;
                }
                else
                {
                    diff = max + min;
```

```
                        if(diff < 0) diff = -diff;
                        sum = max - min;

                        if(diff > diff_max)
                        {
                            state = 0; break;
                        }
                        if(sum < sum_min)
                        {
                            state = 0; break;
                        }
                        if(sum > sum_max)
                        {
                            state = 0; break;
                        }
                        x = (stripe[area][y_max] + stripe[area][y_max-15]) >> 1;
                        for(i = y_max; i >= y_max-15; i--)
                        {
                            if(stripe[area][i] <= x)
                            {
                                helpvar = numb_seam_elem[area]++;
                                seam_elem[area][helpvar].y = (float)i;
                                seam_elem[area][helpvar].x = (float)x;
                                state = 0; break;
                            }
                        }
                        break;
                    }
                }
            }
        }
    }
}



coordinate_transf_module()
/*-------------------------------------------------------------------------*
   function         coordinate_transf_module()
   version          1.0
   date             nov 1989
   author           Werner de Deckere
   institute        Eindhoven University of Technology
                    Department of Electrical Engineering
   group            Measurement and Control
   address          P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

   parameters (+ = entry parameter, - = exit parameter)
      none

   return code
      none

   called subroutines library
      empty

   subroutine description
      Coordinate transformation of the seam element coordinates from
      the sensor coordinate system to the world coordinate system.

   declarations
      none

 *-------------------------------------------------------------------------*/
{
    for(area = 0; area < numb_areas; area++)
    {
        for(j = 0; j < numb_seam_elem[area]; j++)
        {
            seam_elem[area][j].z =
                origin_z +
                nu*(stripe_z[area] + z_adjust[area]*seam_elem[area][j].x);
            seam_elem[area][j].x =
                origin_x +
                labda*((float)stripe_x[area] + seam_elem[area][j].x);
            seam_elem[area][j].y =
                origin_y +
                mu*seam_elem[area][j].y;
        }
    }
}



send_trigger_scope_pulse()
/*-------------------------------------------------------------------------*
   function         send_trigger_scope_pulse()
   version          1.0
   date             nov 1989
   author           Werner de Deckere
   institute        Eindhoven University of Technology
                    Department of Electrical Engineering
   group            Measurement and Control
   address          P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

   parameters (+ = entry parameter, - = exit parameter)
      none

   return code
      none

   called subroutines library
      empty

   subroutine description
      Send 4000 pulses to the SCOPE_TRIGGER port and then execute 4000
      nop instructions (no operation).
      The execution time of this function is 8004*125 nsec.

   declarations
      none

 *-------------------------------------------------------------------------*/
{
#pragma ADSP2100
.external SCOPE_TRIGGER;
            cntr = 4000;              { send 4000 pulses to  }
            do scopeloop1 until ce;   { SCOPE_TRIGGER port   }
scopeloop1: dm(SCOPE_TRIGGER) = si;
            cntr = 4000;              { execute 4000 nop     }
            do scopeloop2 until ce;   { instructions         }
scopeloop2: nop;
#pragma ADSP2100
}
```

```
*-------------------------------------------------------------------------*

  Source file name: adsp_fs.hlp
  Author          : Werner de Deckere
  Date            : Januari 1990

  Eindhoven University of Technology
  Faculty of Electrotechnical Engineering
  Department Measurement and Control
  P.O. Box 513, 5600 MB Eindhoven, The Netherlands

*-------------------------------------------------------------------------*


Variables and function descriptions
-----------------------------------


Constants
---------
  TABLE_SIZE            length of the pixel table array
  PIXELS_PER_SCANLINE   number of pixels on a scanline
  MAX_NUMB_AREAS        maximum number of areas in the screen
  X_MIN                 minimum x coordinate: distribution_module()
  X_MAX                 maximum x coordinate: distribution module()
  Y_MAX                 maximum y coordinate


Data structures
---------------
  Pixel table data structure
  --------------------------
    struct field { x, y }
    struct field pixel_table[TABLE_SIZE]
    struct field *pixptr          Pointer, points to empty place in the pixel
                                  table
    struct field *pixptr_max      Pointer, points to the end of the pixel table
    struct field *distr_ptr       Pointer, points to the element of the pixel
                                  table, that is to be processed.
    int numb_pixels               Number of pixels in the pixel table.


  Stripe and filtered stripe (convol) data structures
  ---------------------------------------------------
    int stripe[MAX_NUMB_AREAS][Y_MAX]  stripes
    int convol[MAX_NUMB_AREAS][Y_MAX]  filtered stripes
    int *ptr, *help_ptr, *stripe_ptr, *convol_ptr
                                  Pointers to the stripe and convol data struct.
    int length[MAX_NUMB_AREAS]    The stripe's length
    int x_prev[MAX_NUMB_AREAS]    the stripe's previous x coordinate
    int y_prev[MAX_NUMB_AREAS]    the stripe's previous y coordinate
    int counter[MAX_NUMB_AREAS]   refer to distibution_module()


  Seam element data structure
  ---------------------------
    struct seam_elem_record { x, y, z }
    struct seam_elem_record seam_elem[MAX_NUMB_AREAS][16]
                        seam element list
    struct seam_elem_record *seam_elem_ptr
                        pointer to the seam element data structure
    int numb_seam_elem[MAX_NUMB_AREAS]
                        number of seam elements


Process parameters
------------------
  int displacement;     refer to function: insert_new_pixel()
  int smooth_factor;    refer to function: insert_new_pixel()
  int stripe_type;      refer to function: insert_new_pixel()
                          1: stripe type a
                         -1: stripe type b

  int threshold;        refer to function: detection_module()
  long diff_max;        refer to function: detection_module()
  long sum_min;         refer to function: detection_module()
  long sum_max;         refer to function: detection_module()

  int direction;        refer to function: detection_module()
                          1: stripe goes to the right
                         -1: stripes goes to the left
  int numb_areas;       number_of_areas: numb_areas <= MAX_NUMB_AREAS


Variables for area definition
-----------------------------
  int numb_areas        (process parameter)
  int stripe_x[MAX_NUMB_AREAS];
                        Estimated position (x coordinate) of the stripes
                        of light refer also to stripe_z (coordinate
                        transformation module)

  int area_bound[MAX_NUMB_AREAS];
                        Boundaries of the area. These are calculated in
                        the function initialization()
  int area,
  int *area_ptr,
  int bound;            boundary

Variables for the distribution module
-------------------------------------
  int displacement      (process parameter)
  int smooth_factor     (process parameter)


Variables for the filter module
-------------------------------
  int pm dog_kernel[16] = { -1, -3, -3, -1,  0,  1,  3,  4,
                            4,  3,  1,  0, -1, -3, -3, -1};
                        Filter coefficients of the discrete DOG filter.
                        Note that the array dog_kernel is located in the
                        data section of the program memory.


Variables for detection module
------------------------------
  int threshold         (process parameter)
  int diff_max          (process parameter)
  int sum_min           (process parameter)
  int sum_max           (process parameter)
  int state             program state
              state ==  0: nothing found in the stripe-signal yet.
                        1: low going pulse found, going to find maximum.
                        2: maximum found, going to find minimum.
                        3: step response complex found, now find (x,y) of the
                           overlap seam and store it in seam_elem data
                           structure.
  int min, max          minimum and maximum
  int sum, diff         sum and difference
  int y_max             y coordinate where maximum occurs
```

```
/*-------------------------------------------------------------*
  variables for coordinate transformation module:
 *-------------------------------------------------------------*/
  float origin_x        camera's position: x-coordinate
  float origin_y                          y-coordinate
  float origin_z                          z-coordinate
  float labda_inv       element of the scaling matrix F: 1/labda
  float mu_inv                                           1/mu
  float nu_inv                                           1/nu
  float stripe_z[MAX_NUMB_AREAS]
                        z-coordinate of the reference stripe (stripe_x)
  float z_adjust[MAX_NUMB_AREAS]
                        Dependent on the positioning of the projector
                        with regard to the camera; z_adjust is equal to
                        1 devided by tan(alpha). Alpha is the angle
                        between the axis of the pjector and the axis of
                        the camera.


Some other global variables
---------------------------
  int x_coord, y_coord;
  int l, h;
  int i, j;
  int x, y;
  int x_previous, y_previous;
  int helpvar, k;
  int t;


global variables defined in the source file: header.dsp
-------------------------------------------------------
  int phase             program phase:
              phase == 0: Start phase; wait for start pulse on input int2.
                       1: Start pulse detected; wait for first vsync pulse
                          on input int1.
                       2: First vsync pulse detected; initialise variables.
                       3: Wait for second vsync pulse on input int1. If the
                          service routine 0 is not executed, the pixels, that
                          are stored in the pixel table untill then, are
                          distributed over the areas.
                       4: Second vsync pulse detected. Finish the distribution
                          of the pixels stored in the pixel table.
                       5: Perform function add_tails().
                       6: Perform filter operation: filter_module()
                       7: Perform function detection_module().
                       8: Perform coordinate transformation:
                          coordinate_transf_module()
  int errorcode         If an error occurs the corresponding error number is
                        stored in the variable errorcode:
          errorcode == 0: No errors.
                       1: To many pixels in the videosignal. Pixel table
                          to small to store all pixels.
                       2: Undefined phase detected in interrupt service
                          routine number 1.
  int count             Number of interrupts detected.
  int count_max         Level interrupt is accepted when the contents of the
                        variable count exceeds count_max.
  int store0, store1, store2, store3, store4
                        Variables for temporary storage.
```

Functions

```
-------------------------------------------------------------------------------
  function          serv_rout0()
  version           1.0
  source file       header.dsp
  date              nov 1989
  author            Werner de Deckere
  institute         Eindhoven University of Technology
                    Department of Electrical Engineering
  group             Measurement and Control
  address           P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

  parameters (+ = entry parameter, - = exit parameter)
     none

  return code
     none

  called subroutines library
     none

  subroutine description
     If the pixel table is not filled yet, then read the contents of the
     XREG and YREG registers and store in the pixel table

  declarations
     none

-------------------------------------------------------------------------------
  function          serv_rout1()
  version           1.0
  source file       header.dsp
  date              nov 1989
  author            Werner de Deckere
  institute         Eindhoven University of Technology
                    Department of Electrical Engineering
  group             Measurement and Control
  address           P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

  parameters (+ = entry parameter, - = exit parameter)
     none

  return code
     none

  called subroutines library
     none

  subroutine description
     If count exceeds count_max and phase is equal to 1 then let phase be
     equal to 2.
     If count exceeds count_max and phase is equal to 3 then let phase be
     equal to 4.

  declarations
     none

-------------------------------------------------------------------------------
  function          serv_rout2()
  version           1.0
  source file       header.dsp
  date              nov 1989
  author            Werner de Deckere
```

```
institute       Eindhoven University of Technology
                Department of Electrical Engineering
group           Measurement and Control
address         P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

parameters (+ = entry parameter, - = exit parameter)
   none

return code
   none

called subroutines library
   none

subroutine description
   If count exceeds count_max then let phase be equal to 1.

declarations
   none
```

```
function        main()
version         1.0
source file     adsp_fs.c
date            nov 1989
author          Werner de Deckere
institute       Eindhoven University of Technology
                Department of Electrical Engineering
group           Measurement and Control
address         P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

parameters (+ = entry parameter, - = exit parameter)
   none

return code
   none

called subroutines library
   initialization(), insert_new_pixel(), filter_module(),
   detection_module(), coord_transf_module()

subroutine description
   This function is the main function of this seam finding algorithm.
   First in each area a mathematical function is generated using the
   elements in the pixel table. The function definitions are stored
   in the stripe data structure.
   Second the stripes can be convoluated with the DOG-filter kernel.
   Third the pattern recognition is done.
   And fourth the resulting coordinates are transformated from the
   sensor coordinate system to the world coordinate system.

declarations
   none
```

```
function        initialization()
version         1.0
source file     adsp_fs.c
date            nov 1989
author          Werner de Deckere
institute       Eindhoven University of Technology
                Department of Electrical Engineering
group           Measurement and Control
address         P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

parameters (+ = entry parameter, - = exit parameter)
   none

return code
   none

called subroutines library
   empty

subroutine description
   In this function a number of variables is initialised

declarations
   none
```

```
function        insert_new_pixel()
version         1.0
source file     adsp_fs.c
date            nov 1989
author          Werner de Deckere
institute       Eindhoven University of Technology
                Department of Electrical Engineering
group           Measurement and Control
address         P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

parameters (+ = entry parameter, - = exit parameter)
   none

return code
   none

called subroutines library
   empty

subroutine description
   In this function one element of the pixel table is considered.
   First the area number is determinated. Second the element is
   added to the stripe of the determinated area.
   If the difference between the x_coordinate of the element and the
   x_coordinate its predecessor in the stripe data structure is larger
   than the variable displacement then the element is not added to the
   stripe data structure. Only when more than a certain number of these
   elements (specified in the variable smooth_factor) are found then
   the element is added to the stripe data structure. The effect is
   that sudden large changes in the description of the stripe are
   removed.

declarations
   none
```

```
function        add_tails()
version         1.0
source file     adsp_fs.c
date            nov 1989
author          Werner de Deckere
institute       Eindhoven University of Technology
                Department of Electrical Engineering
group           Measurement and Control
address         P.O. Box 513, 5600 MB  Eindhoven, the Netherlands

parameters (+ = entry parameter, - = exit parameter)
   none

return code
   none
```

called subroutines library
empty

subroutine description
After the distribution is done, the stripes are undefined in the part from the last pixel till the end. The function add_tails fills this part with the value equals to the x coordinate of the last pixel in the stripe array.

declarations
none

---

function filter_module()
version 1.0
source file adsp_fs.c
date nov 1989
author Werner de Deckere
institute Eindhoven University of Technology
Department of Electrical Engineering
group Measurement and Control
address P.O. Box 513, 5600 MB Eindhoven, the Netherlands

parameters (+ = entry parameter, - = exit parameter)
none

return code
none

called subroutines library
empty

subroutine description
One element (selected by variables area) of each stripe element data structure is convuated with the DOG filter kernel.
The filter constants are stored in the dog_kernel[] array.

declarations
none

---

function detection_module()
version 1.0
source file adsp_fs.c
date nov 1989
author Werner de Deckere
institute Eindhoven University of Technology
Department of Electrical Engineering
group Measurement and Control
address P.O. Box 513, 5600 MB Eindhoven, the Netherlands

parameters (+ = entry parameter, - = exit parameter)
none

return code
none

called subroutines library
empty

subroutine description
This function tries to find the response in the convol data structure that is similar to the DOG-filter step response.
The parameter direction determines whether the function has to look for positive or negative steps (positive or negative overlap seams).
After a place is found where the response is similar to the DOG-filter step response, the function is going to look for the position of the overlap seam.

declarations
none

---

function coordinate_transf_module()
version 1.0
source file adsp_fs.c
date nov 1989
author Werner de Deckere
institute Eindhoven University of Technology
Department of Electrical Engineering
group Measurement and Control
address P.O. Box 513, 5600 MB Eindhoven, the Netherlands

parameters (+ = entry parameter, - = exit parameter)
none

return code
none

called subroutines library
empty

subroutine description
Coordinate transformation of the seam element coordinates from the sensor coordinate system to the world coordinate system.

declarations
none

---

function send_trigger_scope_pulse()
version 1.0
source file adsp_fs.c
date nov 1989
author Werner de Deckere
institute Eindhoven University of Technology
Department of Electrical Engineering
group Measurement and Control
address P.O. Box 513, 5600 MB Eindhoven, the Netherlands

parameters (+ = entry parameter, - = exit parameter)
none

return code
none

called subroutines library
empty

subroutine description
Send 4000 pulses to the SCOPE_TRIGGER port and then execute 4000 nop instructions (no operation).
The execution time of this function is 8004*125 nsec.

declarations
none

```
@echo off
rem filename: m.bat (macro)
rem author  : Werner de Deckere
rem date    : december 1989

echo Macro for using adsp-2100 cross-software
echo usage: m     :assemble header.dsp, compile adsp_fs.c and invoke linker
echo        m a   :assemble header.dsp and invoke linker
echo        m c   :compile adsp_fs.c and invoke linker
echo *

del errors

if "%1"=="c" goto label1

echo assembling...
echo assemble-errors: >>errors
dsppa -c header >>errors

if "%1"=="a" goto label2

:label1
echo compiling...
echo compile-errors: >>errors
cc2100 adsp_fs -m >>errors

:label2
echo link-errors: >>errors
echo linking...
dsppl header adsp_fs -a adsp_fs -c -e adsp_fs -g -x >>errors

type errors
```

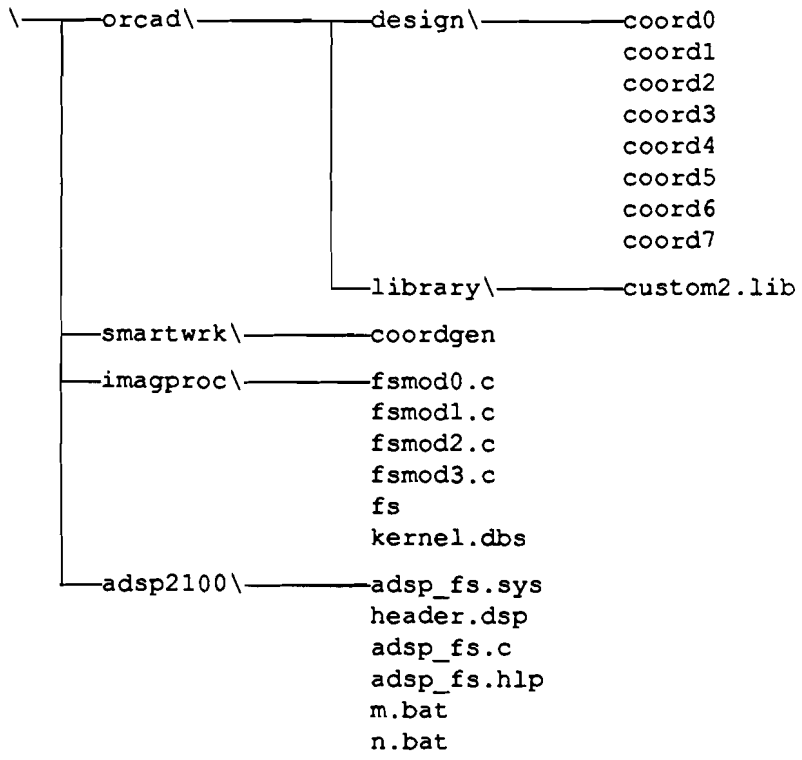
```
@echo off
rem macro for invoking norton editor
rem author   : Werner de Deckere
rem date     :   december 1989
rem filename: n.bat
echo usage: n [a,c,s,m,l] :edit files with extensions: dsp,c,sys,map,lst
if "%1" == "a" ne header.dsp
if "%1" == "c" ne adsp_fs.c
if "%1" == "s" ne adsp_fs.sys
if "%1" == "m" ne adsp_fs.map
if "%1" == "l" ne adsp_fs.lst
```

Appendix L: Pattern recognition software diskette

Pattern recognition software diskette:

floppy disk:	$5^1/_4$ inch
format:	360 Kbyte
label:	Pattern recognition software, Werner de Deckere

Directories and files on the pattern recognition software diskette:



D E T O E P A S S I N G

Dick van Eijk

Op tafel ligt een stuk ijzer met een groeve erin. Een projectortje werpt er lichtstreepjes op en een televisiecamera toont het tafereel op een kleurenmonitor. Er omheen staan computers, apparaten en jongelieden in trui en spijkerbroek. Dit is technisch onderzoek.
De bedoeling is om een robotsysteem te bouwen dat snel een naad kan volgen. Naadvolgsystemen worden in de industrie wel toegepast, maar alleen voor trage processen zoals lassen. Lassen gaat met een snelheid van enkele centimeters per seconde. Maar lijm spuiten gaat veel sneller. Systemen die dat kunnen bestaan nog niet.
Bij de vakgroep 'Meten en Regelen' van de Technische Universiteit Eindhoven, onder leiding van prof.ir. F.J. Kylstra, wordt zo'n snelle naadvolger ontwikkeld. Twee onderzoekers, drie technici en een wisselend aantal studenten hebben er dagwerk aan. Het project wordt gesteund door de Stichting voor de Technische Wetenschappen (STW).
Het is niet zo'n probleem een computer aan de hand van beeldinformatie een naad laten herkennen. Uit de literatuur zijn tientallen methoden bekend. Het probleem is om het snel te doen. Daartoe moet de grote hoeveelheid informatie die het beeld van een televisiecamera bevat worden teruggebracht tot zijn essentie.
De eerste stap is de driedimensionale werkelijkheid terug te brengen tot een tweedimensionaal beeld. De gleuf of naad geeft echter diepte-informatie en die moet behouden blijven. Dit wordt opgelost door met een projector dunne lijntjes over het object te projecteren. Bij naden en gleuven maken de lijntjes een V- of U-vormige bocht. Op het televisiebeeld zijn die V-tjes goed te zien voor een mens. Voor een computer

De volgende stap in de datareductie is alle grijstinten uit het beeld te halen. Idealiter moeten alleen de geprojecteerde lijntjes als wit overblijven, terwijl de rest van het beeld zwart wordt. Een druk op de knop laat zien dat het inderdaad gaat, maar niet feilloos: hier en daar staan nog andere witte streepjes in beeld. De streepjes, zowel de 'echte' als de andere, zijn enige millimeters dik. Met een beeldbewerkingsmethode die *skeletteren* heet, worden alle rafels er automatisch afgehaald. Over blijven mooie dunne lijntjes.
'Nu kan de software de V-vormen gemakkelijker herkennen', zegt onderzoeker ir. Ruud van Vliet. 'Hij zoekt alles wat op een V lijkt op en verbindt zoveel mogelijk van zulke plaatsen door een vloeiende lijn. Dat is de naad.' Tot zover wat het onderzoek de eerste twee jaar heeft opgeleverd. In principe kon uit een beeld met lichtstreepjes de naad worden teruggevonden. De computer deed er alleen twintig seconden over. Dat duurt veel te lang.

Inmiddels is een nieuw beeldverwerkingssysteem in gebruik genomen. Dit kan deze dingen veel sneller, het nieuwe systeem vindt de naad al in tweehonderd milliseconden. Of dat snel genoeg is, hangt van de robot af die eraan komt te hangen. De onderzoekers willen de tijd terugbrengen tot vijftig milliseconden.
Volvo Car heeft als potentieel gebruiker enige belangrijke aanwijzingen gegeven. Zo wordt in de praktijk een werkstuk altijd in een bepaalde positie gebracht voordat er lijm op wordt aangebracht. Met deze extra voorkennis kan het te analyseren oppervlak worden teruggebracht tot vijf bij vijf centimeter. Dat scheelt flink wat rekentijd.
Het dure beeldverwerkingssysteem is nog te ingewikkeld voor een industriële omgeving. De onderzoekers streven ernaar zoveel mogelijk van hun vindingen in harde (analo-

Ruud van Vliet (rechts) projecteert lichtstreepjes op een autoportier. Op de monitor is te zien dat ze een knik vertonen op de naad.

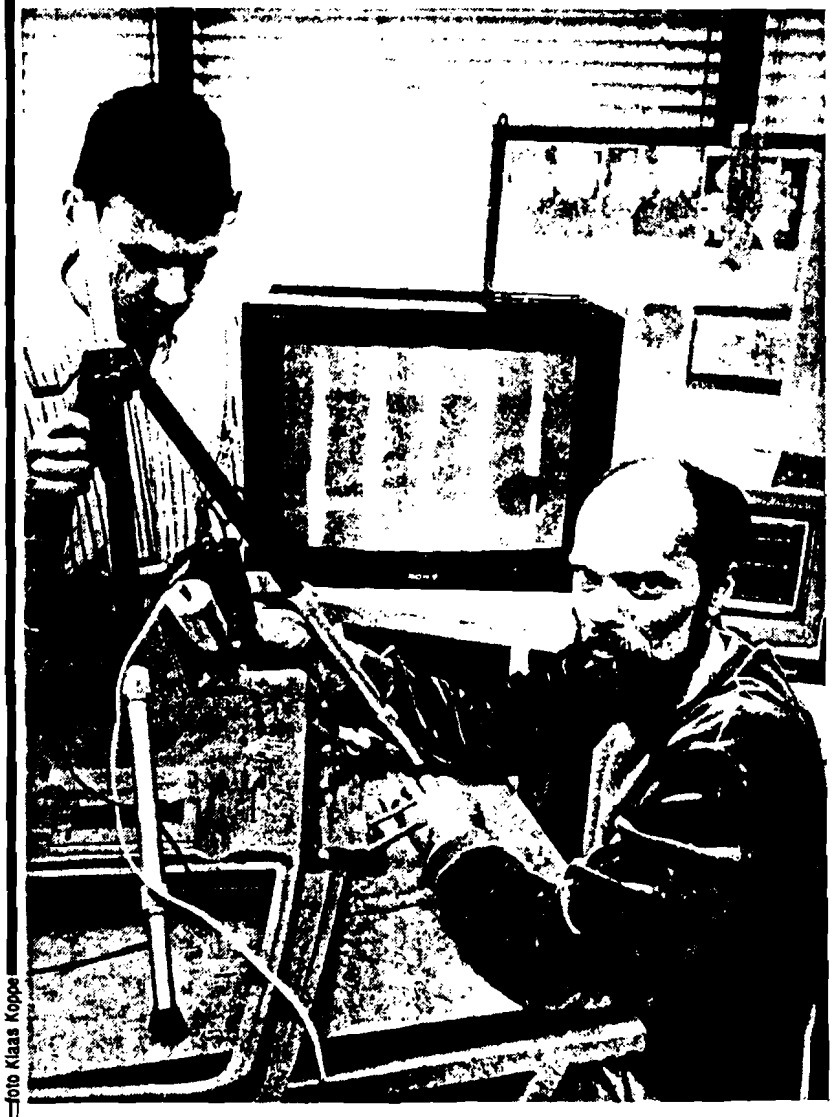


foto Klaas Koppe

'Hij zoekt alles wat op een V lijkt op en verbindt die plaatsen door een

grammeerbare computers. Een van de vijf afstudeerders is nu bezig een filter te ontwerpen om per lijn in het televisiebeeld de helderste punten eruit te halen. Die punten vormen immers samen de lijntjes die de projector op het object heeft geworpen.
Als de computer of later de elektronica eenmaal de coördinaten van de naad heeft gevonden, moet de robotarm hem nog volgen. Van Vliet: 'We zouden de rest aan de robotjongens kunnen overlaten. Maar zij zijn bezig met andere, voor ons doel voorlopig te ingewikkelde robots. Dus hebben we zelf een eenvoudige maar snelle robot gebouwd.'
Het is een zogeheten *portaalrobot*, op het oog een miniatuuruitvoering van een loopkatkraan. De besturing hiervan is overigens nog een apart probleem. Gewoonlijk worden robots van punt naar punt gestuurd. Bij lijm spuiten gaat dat niet, want dan komen er klodders op de omdraaipunten — de arm moet in een vloeiende, gelijkmatige beweging de juiste weg volgen. In de literatuur zijn hiervoor wel methoden bekend. Een andere afstudeerder is nu aan het uitzoeken welke methode in dit geval het geschiktst is.
Dankzij de inzet van afstudeerders en stagiaires is het nog steeds mogelijk om in november een werkend prototype van het hele systeem klaar te hebben — van camera tot en met robot. De weg naar de fabriek is dan niet zo lang meer. Van Vliet: 'Met goed management is het met een jaar toe te passen, mits er voldoende geld is. Zo'n robot kost alleen al 150.000 gulden.'

Alle projecten van de serie 'De Toepassing' worden gesubsidieerd door de Stichting Technische Wetenschappen (STW) te Utrecht.