Simplifications in Visual Servoing

Nielsen, Lars

1985

Document Version:
Publisher's PDF, also known as Version of record

Simplifications in Visual Servoing

Lars Nielsen

Lund 1985

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
DOCTORAL DISSERTATION

Date of issue
September 1985

Document Number
CODEN: LUTFD2/(TFRT-1027)/1-153/(1985)

Author(s)
Lars Nielsen

Supervisor
Karl Johan Åström

Sponsoring organisation
The Swedish Board of Technical Development (STU)

Title and subtitle
Simplifications in Visual Servoing

Abstract

Control based on image information offers great possibilities for advanced automation both in robot motion control and in the process industry. Compared to traditional feedback control systems the measurement is multi-dimensional and complex. There exists not yet an analytical or numerical formulation which captures the essential properties. Here a visual servo experiment has been developed, and parts of the problem have been isolated for further study.

A flexible system for study of solution principles has been obtained by extending a standard computer system with commercially available instrumentation. A robot moving on the floor is controlled by image feedback. It is indicated how it could be applied in flexible storehouses. The man-robot interface uses color graphics entered by pointing in real images of the working scene of the robot to simplify robot motion programming. The main idea is that the graphics is automatically translated to process characteristics.

Marking symbols with perspective invariant descriptions are presented. The problem formulation is new and so are the results and the analysis. The existence of perspective invariants based on areas is a key contribution. The constructed symbols can be used for robot marking or as signposts.

An analytical solution to the optimal trade-off between sampling and quantization under the constraint of a fixed number of bits for image representation is presented. The solution is verified experimentally. It also explains earlier subjective tests.

Key words
Image processing, Vision, Feedback control, Robotics, Perspective invariants, Image digitization.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
153

Recipient's notes

Security classification

Preface

This work has been carried out at the Department of Automatic Control in Lund. I would like to thank all my colleagues for the time past and for a stimulating atmosphere, where you feel free and encouraged to work on problems of your own choice. The following persons are particularly mentioned for their contributions to my work.

I would like to thank my two main supervisors. Karl Johan Åström for his encouraging support and guidance throughout the work. He has continuously helped me planning the work, and he has also picked up a lot of different information and contacts in my own field and fields related to it. Per Hagander has a persistent will to understand and discuss a great variety of subjects. I am grateful to him for spending a lot of time in discussions with me.

During different parts of my work I have collaborated with different people. The image laboratory was developed together with Hilding Elmqvist and Tommy Essebo. Others have also contributed to the laboratory. Bo Nilsson did the Multibus-Unibus interface. Gunnar Sandin and Tord Wullt interfaced the Turtle to the Vax and made touch sensing experiments as their Master thesis. Leif Andersson and Rolf Braun have contributed with both guidance and explicit help in different hardware and software matters.

The visual servo example has been coded almost entirely by Kenneth Johansson during his Master thesis and during voluntary job he did after his thesis. The work on optimal digitization was done mainly during a visit of E.I. Jury in Lund the summer 1983. Beside the explicit work he provided an early introduction into a number of theoretical problems, and he also discussed possible outlines of the thesis. The research efforts by E.I. Jury are partially supported by NSF grant ECS-8116847. His visit was also supported by the guest professor program of the Swedish Board of Technical Development (STU). STU has since 1982 supported my project "Control Based on Image Information" (STU-82-3429). The administration by Gustaf Olsson and Eva Schildt greatly simplifies the work of being responsible for such a project.

I am indebted to Björn Wittenmark and Sven Erik Mattsson for their criticism and suggested improvements of the manuscript. Many thanks also to Agneta

Tuszynski and Eva Dagnegård for their help with the typing and further preparation of the manuscript, and to Britt-Marie Carlsson and Doris Nilsson for careful preparation of the figures.

Last but not least I wish to thank my wife Ingrid for her support and understanding.

Lund the summer 1985
Lars Nielsen

Contents

1. Introduction

The development in image technology is significant and exciting. Video cameras can operate in a variety of light conditions due to improved sensitivity and automatic aperture control. Fast AD-converters and cheap memories are available as plug-in cards for image grabbing, storing, and presentation. Special purpose image processors are commercially available. Custom VLSI design provides means to implement more sophisticated parallel algorithms to be executed in real time.

Applications

Research on the applications of this technology is of significant interest for the field of automatic control. This thesis studies control based on image information, or synonymous visual servoing or image feedback systems. Potential applications of the technique include.

* Control of an overhead crane. A camera can measure the load movements, in order to control the motion of the crane.

* Flexible manufacturing systems (FMS). Supervision and control of systems including manipulators, conveyor belts, trucks etc.

* Control of automatic guided vehicles (AGV). Camera(s) may be mounted high enough to be able to overlook the movements. There may be additional cameras on the vehicles.

* Robot manipulators with computer vision. Typical problems include assembly and cooperating robots.

* Maneuvering of unmanned underwater vehicles using TV cameras. This could mean inspection of the inside of pipelines.

These examples all refer to motion control (Brady et al, 1982; Moravec, 1977). Other categories of problems appear in the process industry where there are potential applications for new sensors based on detection of edges, shapes, color shifts etc. Some examples are:

* Firing in bark ovens with sloping grates. It is important to control the air flows in different sections. The control objective is good firing. Temperature measurements are used today. These are only indirect measures of good firing. A visual servo system looking at the flames offers new possibilities. The control system maintains properties typical for good firing. Also uniformity over an area may be controlled.

* The position of the wet line on a paper machine conveys important information on the drainage on the wire. The wet line can be observed using a vision system. The information obtained may be used adjust the slice screws or other machine settings.

* Semi-insulating boules of GaAs are grown for production of logic circuits. It is desirable to control the growth of the boule diameter using video-images. It may also be possible to obtain a feedback from the spatial distribution of different defects in wafers. The control signals are growth process parameters (Silverberg et al, 1985).

Visual servoing

A control system is based on sensory feedback from the process variables. Process variables are traditionally measured by sensors for physical, chemical, or biological properties. The nature of the measurement is quite different when a camera is used. Basically, the measurement is multi-dimensional. The image may be complex, and it is in general not well understood how the relevant information should be extracted. The control must be based on a pattern in the image, a position of a discontinuity in the image, recognition of a shape etc. The main objective of visual servoing is good control performance. Good reconstruction of measurements (the images) are of secondary interest. Furthermore, if the measurements (the images) are poor at some time instants, it may nevertheless be

possible to get good estimates by using the time sequence of measurements. Therefore the goal for image processing in visual servoing differs from the goal for the image processing in image enhancement, image reconstruction, or image coding, see e.g. Rosenfeld and Kak (1982), Pavlidis (1982), and Pratt (1978). Feedback to the measuring equipment is another interesting feature of visual servoing. The properties of the image obtained can be influenced significantly by changing the camera aperture, lightening, sensitivity, thresholds, and zoom. This situation does not occur with conventional sensors.

The goal of visual servoing also differs from the type of image analysis where the goal is to get a detailed description of a scene, or to get computational models of the human vision system (Artificial Intelligence, 1981; Marr, 1982). In visual servoing the concept of sufficient information for control is used. This information may be extracted without using all details of the complete image. Assume e.g. that the position of an object is to be found. It may then be sufficient to use low resolution, to search for a characteristic feature, or to analyse interesting parts of the image only. The search for such simplifications is a main theme in visual servoing. It is also in line with the tradition in control engineering to establish control principles, which is a broad indication of how a process should be controlled (Åström and Wittenmark, 1984, Chapter 7).

The research on visual servoing was started at the Department of Automatic Control in the spring of 1982. Discussions were held with foreign theoretical researchers, research groups in Sweden, and Swedish industry. A research contract was obtained in the summer of 1982 from the Swedish Board of Technical Development (STU). The idea has been to concentrate the research on principle solutions to visual servoing. Clear images with good contrast have been used. Image description is nontrivial even with good images. The connection between image processing and control has been the primary issue. Search for control principles and simplifications has been a main objective. Some problems of special interest are three dimensional (3-D) aspects in realistic scenes. The hardware status and guesses about future hardware development has guided the selection of problems.

This thesis

The theory of visual servoing has not developed so far that the closed loop performance can be judged only by analysis of theoretical models or simulations. A prototype system therefore had to be built, and significant experiments with the system have to be tried. The philosophy has been to develop a flexible laboratory at a reasonable cost. Simple robots have been used. The experimental set up is described in Chapter 2. It is implemented by extending a standard computer system with readily available instrumentation. The laboratory has been used in the experimental studies of the visual servo example described in Chapters 3 and 6. A robot moving on the floor is controlled by image feedback. The software includes image processing, image interpretation, perspective corrections, feedback path control, route planning, obstacle avoidance and collision handling using touch sensors. Furthermore the visual servo contains a man-robot interface based on menu driven, interactive computer graphics combined with images of the working scene of the robot. Interesting problems have been singled out and studied in detail, and the results are used as building parts in constructing the closed loop visual servo. The experimental test bed is flexible. It has given insight into useful principles for the visual servo problem.

An approach to image processing based on edge detection and contour description is reviewed in Chapter 4. Special purpose hardware implementing the methods is commercially available. Given this knowledge we have proceeded to develop algorithms for picture changes as a result of different perspective of objects. The shape and the size of objects vary when viewed in different perspective. There are significant differences compared to inspection of parts on a conveyor belt. The object is not at a known distance from the camera. On a conveyor belt the objects have only a few stable poses, that may all be stored. In 3-D an object has infinitely many poses. In Chapter 5 simplifications obtained when using easily detectable marking symbols is discussed. The formulation and results are based on the general approach to image processing outlined in Chapter 4. The key idea is to use a marking symbol, which has a perspective invariant property. The symbol may then be localized from this property which can be calculated directly from the image independently of the actual pose. Marking symbols, and simple and safe interpretation algorithms are presented in Chapters 5 and 6. Both the idea and the analysis are to my knowledge new contributions.

The amount of image data that must be handled is a general problem. The possibility to represent an image with less data decreases the computational cost. A definition of optimal digitization (quantization and sampling) of 2-D images has been given for the first time. This is presented in Chapter 7. An advantage is that the solution is given in closed form, which clearly points out the dependence on image characteristics. It is experimentally verified that the proposed problem formulation makes sense.

Conclusions drawn from the thesis are given in Chapter 8. Suggestions for future research are also given. It is hoped that the research reported herein will stimulate others both in this university and elsewhere to continue the work.

2. A Visual Servo Laboratory

The development of a visual servo laboratory includes extension of hardware and software development. This chapter presents a system that is built by extending a conventional computer with a raster memory. The computational speed is limited by the speed of the computer, but all aspects of the visual servo problem may in principle be demonstrated.

Some recent technological advances have made it easy to obtain a good system for experimenting with images. Video-cameras with increased sensitivity and automatic control of the aperture are available. Such low cost video cameras may without any adjustments be operated in illuminations varying from outdoor summer season daylight to indoor standard lamps. A raster image memory with simple interfaces can be built from commercially available plug-in boards. The image memory uses standard video-signals to external video equipment such as video-cameras, video-recorders, or monitors. The connection to the computer is via a general bus. It is also easy to work with large amounts of data on a conventional computer either by having enough primary memory or by using the virtual memory technique. It is thus neither necessary to build special hardware nor to make complicated file handling to deal with images. The remaining implementation problems concerning image data may be considered as software engineering. The goal for the software development has been to design and implement a system which is convenient and easy to use. All programming can be made in a high level language.

The laboratory equipment is described in Section 2.1. Section 2.2 treats the low level software for communication between the raster image memory and the Vax computer. The structure of the software and the basic support packages are presented in Section 2.3. A complete user example is presented in Section 2.4. Conclusions are given in Section 2.5.

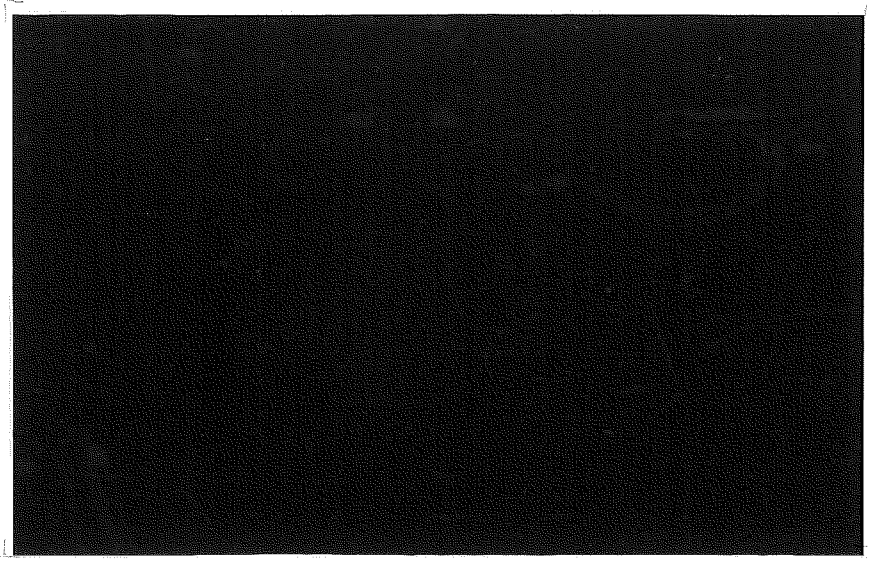


Figure 2.1 The work bench of the visual servo laboratory. The terminal is connected to the Vax. The RGB-monitor presents the content of the image memory. The mouse and the joysticks are used for interactive image handling and for manual control of the vehicles.

2.1 Hardware

The image system is built around a Matrox raster image memory interfaced to a VAX-11/780 via a Unibus-Multibus-interface. The hardware is listed in Table 2.1. A schematic overview of the system is given in Figure 2.2.

The raster image memory

The raster image memory is built from a family of commercially available plug-in boards manufactured by Matrox. The system contains two RGB-Graph/64-4 boards and one VAF-512 board (Matrox manuals No:s 167MO-04-0 and 176MO-01-0). The RGB-Graph boards provides the system with a raster memory of 512 x 512 pixels with 8 bits/pixel, where each board stores 4 bits/pixel. The VAF-512 board is a video input/output processor board. It contains three main functions: a real time video digitizer (frame grabber), a color look-up table, and a high speed vector generator. The boards are controlled via I/O registers. Each RGB-Graph board looks like 16 consecutive 8-bit wide I/O locations from the Multibus. The VAF-512 board looks like 12 such locations. A full description of the programming facilities is found in the Matrox manuals. Together these boards form the image memory. The image memory uses videosignals as input and output

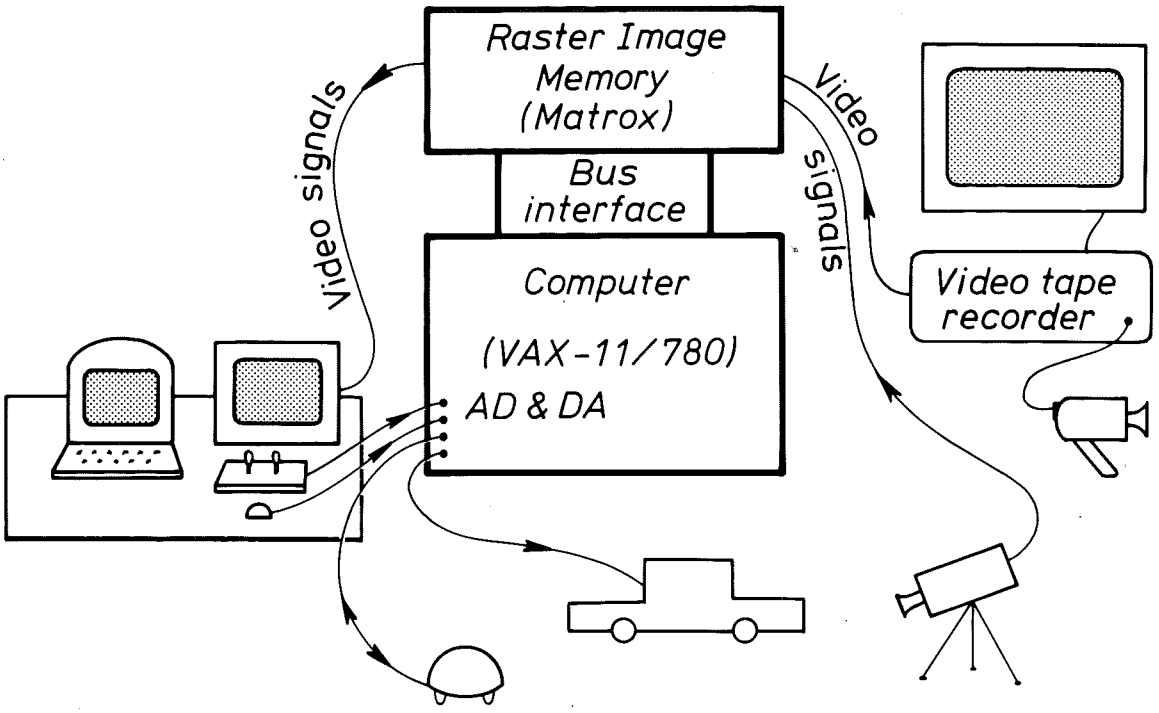


Figure 2.2 A schematic overview of the system. The system consists of different pieces of standard equipment. The interfaces are simple and general. The video camera, videotape-recorder and the monitor are using standard video signals to the image memory. The connection between the image memory and the VAX is via a general bus. The vehicles, the mouse and the joysticks are interfaced to the VAX via the standard AD and DA-converters.

to external video equipment such as video-cameras, video-recorders, or monitors. The timing of the image system may be programmed by software. It can either follow an on-board sync generator or the input video signal via an on board phase-locked loop circuit. The video input hardware and the video output hardware work at the video rate of 25 frames per second. The frame grab commands represent selection of input channel, of repeated image sampling or frozen image, and of gain and offset values in the video quantization. The system has four video input channels. One video input channel is used for the video camera, and another one is used for the video tape recorder. The content of the memory can be read or written down to individual pixels via two address registers (x and y) and a data register. The video output hardware consists of video generators (RGB) which interpret the content of the raster memory via a programmable color look-up table. The color look-up table allows 256 displayed

Computer
1 VAX-11/780

Raster image memory
2 Matrox RGB-Graph/64-4
1 Matrox VAF-512
1 Unibus-Multibus-interface

Video input output
1 Intensa GPC-25 video camera
1 Barco CD 33 HR monitor
1 Telefunken VR530 video tape recorder
1 Hitachi VK-C830E video camera
1 Telefunken PALcolor 1520 TV

Processes
1 Turtle
1 Ilon car

Interactive devices
1 Mouse
2 Joysticks

Table 2.1 Listing of hardware

colors from a palette of 16 million colors. The color look-up table allows presentation of images with immediate pseudo coloring and grey scale transformations. The 8 bits/pixels are physically divided into 4 bits on each RGB-Graph board. Video enable-disable commands can be used to display the content of one board while the other board reads new information.

The video input and output

A high resolution RGB monitor is used to present the video output. The monitor used is shown in Figure 2.1. The video camera used is a black and white Intensa GPC camera (GPC 25 Black and White TV-camera Manual). The camera can be equipped with any tube of the 1" vidicon type having a lens with a C-mount (1"x32 threads/inch). In the laboratory set up the camera is equipped with a 1" newvicon camera tube. A 25 mm lens with automatic aperture control is used. The control is performed locally in the optics. The control signals are not available to the computer. The resolution of the camera is approximately 525 lines and 800 dots per line, the light sensitivity is 1-2 lux, and the automatic light compensation is 1:30000. The VAF-512 board of the image memory provides the master sync pulses of the image memory, the camera and the RGB monitor.

We have also a home video set with a video tape recorder, a color video camera, and a TV. The video tape recorder is connected to the image memory. The camera and the TV are connected to the video tape recorder. The color video camera has a resolution of roughly 250 lines/image, a light sensitivity of 60 lux, and an automatic light compensation of 1:3300. The video set has lower quality than the GPC camera. The video set provides, however, an easy way to handle image information from other places via the exchange of video tapes.

The image memory - Computer connection

The image memory is accessed by programmed I/O. The boards are available for the Multibus, and the VAX operates on a Unibus. A Multibus-Unibus-interface has been developed by Bo Nilsson at the Computer Engineering Department (Nielsen and Elmqvist, 1983). This interface is transparent, which means that the translation from Unibus to Multibus format and vice versa is performed without any time delays. The bus communication bandwidth is approximately 1.6 Mbyte/s.

Utilities

The joysticks and the mouse shown in Figure 2.1 are used for interaction. The mouse has three on/off buttons. The Vax has general purpose AD- and DA-converters. They are used for the signals from the joysticks and the mouse. They are also used for the robot interfaces.

The robots

In general any process may be interfaced to the Vax via its AD- and DA-converters. In our experiments vehicles moving on the floor are used. They are seen in Figure 2.3.

One vehicle is a commercially available toy Turtle marketed by Terrapin, Inc. (Terrapin Turtle assembly manual). The Turtle has a dome which is 30 cm in diameter. It has two separately driven wheels. Each wheel is individually controlled via on/off signals. For each wheel the drive train has an electrical motor and a gear-box. The motor makes 300 revolutions for one revolution of the wheel. The maximum speed is roughly 15 cm/s. The Turtle is controlled via a cord that transfers electrical signals in the range 0-18 V. An interface to the Vax has been developed (Sandin and Wullt, 1982). The basic motion primitives for the

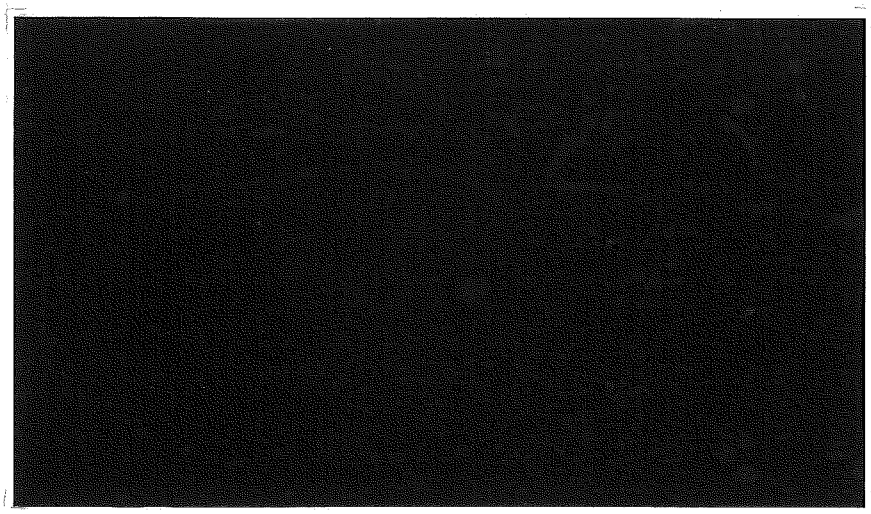


Figure 2.3 Photograph of the two vehicles. The Turtle is seen to the left and the Ilon car is seen to the right.

Turtle are given in Appendix B. The Turtle has touch-sensors. If the dome of the Turtle is tilted due to the contact with an object then the status of four switches indicates one of eight directions. The control system of the Turtle may thus combine image and touch information.

The other vehicle is the Ilon car, which was available at the department of Automatic Control (Åström, 1975). The dimensions are: length 60 cm, width 25 cm, and height 20 cm. The Ilon car has four separately driven Ilon wheels. It allows complete control of the motion in the plane via control of the rotation of the four wheel motors. The Ilon car may thus be moved straight forward, straight sideways or be rotated on the spot. The on-board electronics of the Ilon car is controlled using three analog voltages in the range -10 - 10 V, which are set points for the speed of respectively the forward motion (negative voltage means backward motion), the sideward motion (positive voltage to the right and negative to the left), and the rotation motion (positive voltage to the right and negative to the left).

2.2 The Low Level Software

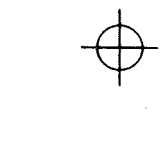
The function of the raster memory is controlled via information passed via its control registers. The low level software is a package containing routines for communication between the raster memory and a program on the Vax. The routines are implemented in assembly language. A complete listing of the Pascal declarations of the routines is given in Appendix A. The main functions are as follows. The registers of the raster memory are mapped into addresses of the Vax virtual memory using normal system services in the operating system for handling the Unibus. The different registers of the raster memory then have addresses within an I/O-page of the Unibus. The procedures MultiGet and MultiSet are general operations for reading and writing on these addresses. The organization of the I/O-page has to be known when using these procedures. The operation codes are given in the Matrox manuals. Figure 2.5 gives an example.

Any operation on the raster memory may be expressed as a sequence of calls of the procedures MultiGet and MultiSet. Numerous procedure calls in the user program is, however, inconvenient and also reduces execution speed. Some time critical and often used communication procedures are therefore implemented in assembly language. These specialized procedures describe functions such as reading (RasterRead) or writing (RasterWrite) an image. The handling of the address and data registers are hidden in the code. An image variable and image size parameters are the only parameters of these procedures (RasterRead, RasterWrite) accessible by the user. More operations on the raster memory are given in the description of the support packages in Section 2.3.

2.3 Software Development Environment

The software tools will now be described. The modularization of the code in packages will be presented, together with the basic support packages. The support packages contain sufficient operations to build user programs without knowing the explicit organization of the hardware.

We have chosen to use the programming language Pascal. Compared to the alternatives of assembly language or FORTRAN, it provides data structures and



recursion. The efficiency of the Pascal compiler with respect to the size of executable code and the execution speed are both roughly the same as for a FORTRAN program. The alternative of using Lisp falls on the low execution speed on the Vax. Irrespective of the choice of language the virtual memory technique eliminates the need for special code or special file handling to declare matrices of the size 512 x 512.

The structure

One could foresee large programs when the project was started. The idea of object oriented programming was therefore adopted. This means that modularized code can be developed so that types, variables, procedures which are logically connected, can also be kept together textually. The modularization in packages makes it easier for several implementors to work together in the same project. The packages are implemented as text files with a special structure. A file handling preprocessor is used to combine the packages into a standard Pascal program. The ideas around the software structure with the preprocessor and the packages originates from Hilding Elmqvist. It has been used in the project "Development of User Oriented Languages and Software Tools for Implementation of Control Systems" (Elmqvist, 1983).

The packages

The packages contain standard Pascal statements and preprocessor keywords. The preprocessor keywords are commands to the preprocessor which divide the file into sections corresponding to the sections of a Pascal program.

The first seven commands of the list in Table 2.2 define the start of a text section of a Pascal program. The command .INIT defines a section which will be placed first in the main program. The command .END requests the preprocessor to neglect the text until the next keyword. The text handling by the preprocessor is general. The preprocessor keywords may thus be used to specify sections of a Pascal program in any order, e.g. to mix .TYPE, .VAR, .PROCEDURE sections. A certain programming style has been adopted. It is maintained by discipline of the users. There is no automatic check. The style is inspired by Ada, where the packages are structured in terms of a specification and a body. The specification contains declarations of what is available outside the package. The body contains

.PROGRAM	.FORWARD
.LABEL	.PROCEDURE
.CONST	.MAIN
.TYPE	.INIT
.VAR	.END
.EXTERN	

Table 2.2 The section keywords.

the implementation. In Pascal, the procedure specification may be separated from the procedure implementation by declaring the procedure forward in a .FORWARD section of the specification part of the package. The code of the procedure is given in the .PROCEDURE section of the body part of the package. The package skeleton in Figure 2.4 illustrates our style of structuring a package. Compare also Appendix A.

The preprocessor

The preprocessor ComPack is a file handling program used to combine packages to form one standard Pascal program. The input to the preprocessor is a list of files of package type (with filename extension .pak). The preprocessor reads the files and splits them into sections according to the section keywords. The output of the preprocessor is nine files of section type (with filename extension .sec), where the sections of each package file have been added to the corresponding section files. A Pascal program named compackprog includes the section files.

Basic support packages

A number of support packages are available. The low level software described in Section 2.2 is one support package. That package specification consists of external declarations of the low level routines. The package body is the assembly language implementation which is put in a library.

There are also a number of support packages, for a core of image handling and for control of the hardware, which are implemented in Pascal. The low level software is used to implement the support packages. The implementation of the procedure SetColorMap in LookUp.pak is shown in Figure 2.5 as an example. It is

```
-------------------------------------------------------------
package Skeleton is
{ An information text. }
-------------------------------------------------------------
.CONST
{ Standard Pascal constant declarations intended to be used
  also outside the package. }
.TYPE
.VAR
.FORWARD
{ The procedures provided by the package. }
.END
-------------------------------------------------------------
package Skeleton body is
.CONST
{ Constants used only internally in the package. }
.TYPE
.VAR
.PROCEDURE
{ Procedures. Both procedures used only internally in
  the package, and the procedure bodies of the
  procedures specified in the FORWARD section of the
  specification part of the package. }
.INIT
{ Initialization section placed first in the main program
  by the preprocessor. }
.END
-------------------------------------------------------------
```

Figure 2.4 A typical package structure.

used to set one entry of the color look-up table. Most procedures of the package LookUp.pak are implemented using SetColorMap. The support packages include sufficient operations to build user programs without knowing the explicit organization of the hardware.

```
procedure SetColorMap{color, r, g, b: integer};
  const VAFaddr = 96;
  begin
  MultiSet(VAFaddr+1, 0);
  MultiSet(VAFaddr+0, color);
  MultiSet(VAFaddr+8, r);
  MultiSet(VAFaddr+0, color);
  MultiSet(VAFaddr+9, b);
  MultiSet(VAFaddr+0, color);
  MultiSet(VAFaddr+10, g);
  end;
```

Figure 2.5 Support packages are available providing routines which relieves the user from knowing the explicit organization of the hardware. The implementation of one such routine is illustrated here. The address, within the I/O-page, of the VAF-512 board is 96. The registers used are 0,1 specifying the color look-up entry and 8,9,10 which are data input registers for red, green, and blue values respectively.

The specification part of the basic Pascal packages in the lab system are listed in Appendix A. They include

* Video input commands (RasterReg.pak)
* Image definitions and in/out operations (ImageDef.pak)
* Color look-up table handling (LookUp.pak)
* AD- and DA-conversion (AdConv.pak,DigConv.pak)
* File handling to save and restore images (SimplFil.pak)

Further abstraction levels may be developed based on the basic packages. A package for handling the mouse and the joysticks is built using ADConv.pak and DigConv.pak. The control of the Turtle and the further levels of image interpretation are presented later. Procedures for handling real time programming in Pascal are available (Nielsen and Elmqvist, 1983). The communication between concurrent processes is based on the rendezvous concept in Ada.

2.4 A User Example

A complete user program is presented as an example. It is a simple program for image handling. It has commands for image input and output, coloring of output, simple pixeloperations, and image convolutions. Both the program and the command procedure which generate the program are listed. These listings illustrate how the support packages and the preprocessor have been used. The extension to more complicated image processing or interaction is only a matter of further software development.

General

The program operates on one image which is obtained from the video camera or from a stored file. The image is presented on the screen. There are commands to manipulate the image. The result of a command is displayed on the screen. The different commands are described in the next paragraph where the user interface is described.

The user interface

The user interface is based on a menu. The menu is shown in the section .MAIN of the code listing. One letter commands are used.

The first line of the menu is E[xit], F[reeze], G[rabCont], S[ave], R[estore]. The Exit command is used to leave the program. The Freeze and GrabCont commands control the input from the video camera. The command GrabCont sets the image memory in the mode for repeated image sampling and the command Freeze stores one image and stops. This image is then available for further processing. The command Save stores the image on a file. The command Restore gets the actual image from a file. Both commands asks for the filename to be used.

The second line of the menu (B[lackWhite], P[seudoColor]) contains two commands for manipulating the color look-up table. With the BlackWhite command the image looks natural. The PseudoColor command assigns some image values to be presented as red, some as blue, etc. The image data itself is not changed, only its presentation.

The third line of the menu gives two commands for pixel operations. The Threshold command results in a bilevel image. All image values above the mid value are set to the maximum value and all other image values are set to the minimum value. The command Invert creates the negative image.

The fourth line of the menu contains commands for convolution with an operator of the size of 3 x 3. The operator may be defined using three different commands. The LowPass command sets the operator to a standard low-pass filtering operator defined in the program code. The High-pass command sets the operator to a high pass filtering operator. The command NewOperator allows the user to enter an arbitrary 3 x 3 operator. The operator is automatically normalized so that the weights sum to one. The actual operator is applied to the image using the command Operate.

The implementation

The Pascal program demo.pak is structured as a package using the section keywords. The program makes use of the support packages Raster.pak, RasterReg.pak, LookUp.pak, ImageDef.pak, and SimplFil.pak. The specification of these packages are given in Appendix A.

The code listing is hopefully close to being selfexplanatory. The variable Operator contains the actual convolution operator. The variable InImage is the actual image presented on the screen, and the variable OutImage is an auxiliary variable. The type "imagetype" is declared as a matrix of bytes in the support package ImageDef.pak and the variable "imagefile" is declared in SimplFil.pak. The interpretation of the commands is performed in the case statement with comchar as tag. It is seen how both Freeze and Restore put the input to the program in InImage, and how InImage is kept as actual image.

Listings

The program (demo.pak) is listed together with the DCL command procedure (demo.com) used to generate the program. In the command procedure the lines starting with a $ is a DCL command and the lines starting with a ! is a comment.

The program development is streamlined by the use of a command procedure. The program is structured into one or more packages. Any procedure from any

support package may be used. The command procedure calls the preprocessor and gives a list of the packages used. The link command invokes the libraries to be used. The program modification cycle is

1. Edit demo.pak
2. @demo
3. run demo

```
! Command procedure to generate the DEMO program

! DCL symbol starting compack followed by
! the list of used packages.
$ ComPack
Raster.pak
RasterReg.pak
LookUp.pak
ImageDef.pak
SimplFil.pak
Demo.pak
! Compile the Pascal program with the name compackprog.
$ pascal/object=demo compackprog
! Link the program. Invoke the library Raster.
$ link/nomap demo,rasteropt/opt
```

```
package DEMO is

-- A demonstration program

-- Author: Lars Nielsen
-- Date: 1983-09-07

------------------------------------------------------------

package DEMO body is

.PROGRAM

program demo(input,output,imagefile);

.TYPE

optype        = array[-1..1,-1..1] of real;

operatortype  = record
  op          : optype;
  opsum       : real;
end;

.VAR

exit                    : boolean;
comchar                 : char;
operator                : operatortype;
InImage, OutImage       : imagetype;    { from ImageDef.pak }

.PROCEDURE

{-----------------------------------------------------------}

function limit(val: integer): integer;
  const minI = 0; maxI = 255;
  begin
  if val > maxI then limit:=maxI
  else if val < minI then limit:=minI
  else limit:=val;
  end;

{-----------------------------------------------------------}

procedure Threshold(level: integer;
                    var InImage, OutImage: imagetype);
  { threshold InImage, put result in OutImage }
  const minx = 1; maxx = 512; miny = 1; maxy = 512;
        minI = 0; maxI = 255;
  var x,y: integer;
  begin
  for y := miny to maxy do for x:= minx to maxx do
  if InImage[y,x] > level then OutImage[y,x] := maxI
  else OutImage[y,x] := minI;
  end;

{-----------------------------------------------------------}

procedure Invert(var Inimage, OutImage: imagetype);
  { Invert InImage, put result in OutImage }
```

```
  const minx = 1; maxx = 512; miny = 1; maxy = 512;
        minI = 0; maxI = 255;
  var x,y: integer;
  begin
  for y := miny to maxy do for x:= minx to maxx do
  OutImage[y,x] := maxI - InImage[y,x];
  end;

{-------------------------------------------------------------}

procedure NewOperator;
  { define new 3 x 3 neighbourhood operator }
  var i,j : integer;
  begin
  with operator do
    repeat
    for j:=-1 to 1 do for i:= -1 to 1 do
      begin
      write('operator[ ',j:2,', ',i:2,'] := ');
      readln(op[ j, i]);
      end;
    opsum:=0;
    for j:=-1 to 1 do for i:= -1 to 1 do
      opsum:=opsum+op[j,i];
    until opsum <> 0;
  end;

{-------------------------------------------------------------}

procedure LowPassOperator;
  begin
  with operator do
    begin
    op[-1,-1] := 1;  op[-1, 0] := 1;  op[-1, 1] := 1;
    op[ 0,-1] := 1;  op[ 0, 0] := 2;  op[ 0, 1] := 1;
    op[ 1,-1] := 1;  op[ 1, 0] := 1;  op[ 1, 1] := 1;
    opsum := 10;
    end;
  end;

{-------------------------------------------------------------}

procedure HighPassOperator;
  begin
  with operator do
    begin
    op[-1,-1] := -1;  op[-1, 0] := -1;  op[-1, 1] := -1;
    op[ 0,-1] := -1;  op[ 0, 0] :=  9;  op[ 0, 1] := -1;
    op[ 1,-1] := -1;  op[ 1, 0] := -1;  op[ 1, 1] := -1;
    opsum := 1;
    end;
  end;

{-------------------------------------------------------------}

procedure Operate(var InImage,OutImage: imagetype);
  { apply operator on InImage, put result in OutImage }
  const minx = 2; maxx = 511; miny = 2; maxy = 511;
```

```
  var x,y,i,j,help: integer;
      temp        : real;
      localop     : optype;
      localopsum  : real;
  begin
  localop:=operator.op;
  localopsum:=operator.opsum;
  for y := miny to maxy do for x:= minx to maxx do
    begin
    temp:=0;
    for j:=-1 to 1 do for i:=-1 to 1 do
      begin
      help:=InImage[y+j,x+i];
      temp:=help*localop[j,i]+temp;
      end;
    OutImage[y,x]:=limit(round(temp/localopsum));
    end;
  end;
{------------------------------------------------------------}
procedure Init;
  var x,y: integer;
  begin
  LowPassOperator;
  for x:= 1 to 512 do for y:= 1 to 512 do
    begin
    InImage[ x, y] := 0;
    OutImage[ x, y] := 0;
    end;
  end;
{------------------------------------------------------------}
.MAIN
Init;
exit := false;
repeat
  writeln;
  writeln('------- Menu -------------------------------');
  writeln;
  writeln('E[xit],F[reeze],G[rabCont],S[ave],R[estore]');
  writeln;
  writeln('B[lackWhite],P[seudoColor]');
  writeln;
  writeln('T[hreshold],I[nvert]');
  writeln;
  writeln('N[ewoperator],O[perate],L[owpass],H[ighpass]');
  writeln;
  write('Command > ');
  readln(comchar);
  if comchar in ['e', 'E', 'f', 'F', 'g', 'G',
                 's', 'S', 'r', 'R',
                 'b', 'B', 'p', 'P',
```


```
                't', 'T', 'i', 'I',
                'n', 'N', 'o', 'O', 'l', 'L', 'h', 'H'] then
    case comchar of
      'e','E' :   exit := true;
      'f','F' :   begin
                  GrabOne;              { from RasterReg.pak }
                  ReadImage(InImage);    { from ImageDef.pak }
                  end;
      'g','G' :   GrabCont;             { from RasterReg.pak }
      's','S' :   SaveFile(InImage);     { from SimplFil.pak }
      'r','R' :   begin
                  RestoreFile(InImage); { from SimplFil.pak }
                  WriteImage(InImage);  { from ImageDef.pak }
                  end;
      'b','B' :   BlackWhite256;          { from LookUp.pak }
      'p','P' :   Pseudo;                 { from LookUp.pak }
      't','T' :   begin
                  Threshold(128,InImage,InImage);
                  WriteImage(InImage);
                  end;
      'i','I' :   begin
                  Invert(InImage, InImage);
                  WriteImage(InImage);
                  end;
      'n','N' :   NewOperator;
      'o','O' :   begin
                  Operate(InImage, OutImage);
                  WriteImage(OutImage);
                  InImage := OutImage;
                  end;
      'l','L' :   LowPassOperator;
      'h','H' :   HighPassOperator;
    end
  else writeln('Error in command');
until exit;

.END
```

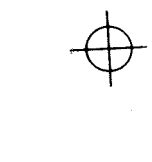
2.5 Conclusions

Our experimental equipment is comfortable and easy to handle. Its speed is limited by the capability of the Vax computer. All solutions of the visual servo problem can in principle be demonstrated with the system.

Three technologies which influence the implementation of an image system have matured to commercialization. The video cameras can be equipped with a high sensitivity vidicon combined with automatic aperture control. This solves the problem of getting good visual quality of the image data, when the illumination changes. A raster image memory with general interfaces (both for the video and the computer) may be built from commercially available plug-in boards. The virtual memory technique of the Vax automatically solves the problem of addressing image data in software.

One approach to the software structuring has been presented. The programming language Pascal is used. A preprocessor was developed to be able to modularize the code while keeping to standards. A programming style inspired by Ada was adopted. The software structure has proved useful for large programs.

Any process may be interfaced to the Vax via the AD- and DA-converters. Two vehicles moving on the floor have been presented. The system has also been used in experiments on inspection of GaAs-wafers (Silverberg et al, 1985), on motion detection in image sequences (Nielsen, 1984b; Jansson ,1985), and on measurement of the position and shape of the ash-line in bark ovens (Dahl, 1985). The latter experiments were performed using the video tape recorder to bring in images from an industrial bark oven.



3. A Visual Servo Example

Robot motion control, flexible manufacturing systems (FMS), and control of automated guided vehicles (AGV) are some examples of industrial problems that can be approached using visual servoing. These problems typically involve image feedback, motion control, planning and replanning, and handling of external events. These tasks will be illustrated in this chapter. It is not clear how these problems should be approached. The research direction here has been to rely on real experiments. The hardware described in Section 2.1 has been kept fixed but the software has developed in different directions during the experiments. Parts of the experiment have been presented earlier (Johansson, 1984; Nielsen, 1984a; Nielsen and Johansson, 1984).

Section 3.1 describes a simple experiment which captures many features of the problem. The objective is to let the Turtle perform a simulated workcycle where its motion is controlled by image feedback. A sequence of images illustrating the behavior of the Turtle is included. The underlying structure of the experiment is extracted in Sections 3.2-3.4. Details are given in Appendix B. A man-robot interface for programming and supervision is presented in Section 3.5. The implementation experiences are given in Section 3.6, and conclusions in Section 3.7. The image processing used is presented in Chapters 4 - 6.

3.1 The Experiment

The Turtle will simulate an automated guided vehicle (AGV) working on the floor in a storehouse. The scenario is as follows. The Turtle moves on the floor. It performs a workcycle, which consists of visits to work stations. The stations are the fundamental sites in the workspace. They symbolize places where a load is collected or delivered, where an assembly task is performed etc. The

requirements on the motion paths between stations are more flexible. The primary concern is that the Turtle reaches the stations. The scenario also includes obstacles. The obstacles are either described by the operator using the man-robot interface or they are detected by the program. The program checks that the motion paths of the Turtle avoid the known obstacles. This includes when the Turtle detects new obstacles during operation. The paths are replanned automatically. If the Turtle is unable to reach a station, due to new obstacles, it calls for help by sending an alarm to the operator.

A typical experiment will be presented here. Figures 3.1 and 3.2 give an overview. The Turtle simulating an AGV is in the lower middle of the image. The triangle in the lower right of the image is a known reference symbol for the image processing. The reference triangle and the two square pieces of paper on the floor symbolize three work stations. They are numbered 1, 2, and 3 in Figure 3.2. Station 1 symbolizes a deposit of pieces of material that should be handled. Station 2 symbolizes a work station where the pieces of material are processed. Station 3 symbolizes a deposit where the processed pieces of material are delivered. The deposit at Station 1 is filled as follows. The Ilon car now and then comes with a pile of pieces of material. The position of the Ilon car is not specified from time to time. The AGV should go to the Ilon car and make a symbolic load transfer. The AGV is supposed to carry the pile to Station 1 and deliver it there. There are also three obstacles in the scene. The obstacles are the bottom of a chair in the upper right of the image, the white box in the upper edge of the image, and the variable-voltage transformer between the stations two and three. The obstacles are put in the scene to illustrate two different aspects. The chair and the white box symbolize permanent effects e.g. machines and will be defined as obstacles using the man-robot interface. They will thus be known to the robot. The variable-voltage transformer will purposely not be defined. It represents an unknown obstacle e.g. something dropped by another AGV. The AGV will collide with it on the way from Station 2 to Station 3.

We will now follow the experiment as it appears on the monitor screen. The scene, the result of the programming, and the resulting action will be presented in connection with the sequence of images numbered 1-36 in Figure 3.4. These are stylized drawings. Some of them have inserted photographs of the screen showing the output of the edge detection within the subimage actually analyzed.

Start-up

Image 1 and Figure 3.2 are the same and show the start position. The man-robot interface is used to enter the tasks to the robot. The result of the programming is shown as graphics which is overlayed the image of the scene. Three stations represented as blue dots and two obstacles represented as closed red polygons with white grid nets have been defined, see image 2 and Figure 3.3. They have been entered by pointing in the image on the monitor screen. The paths between the stations have been calculated by the path finding algorithm. The result is paths avoiding the two known obstacles. The paths have attributes which are shown as colors. On the red paths the AGV is not allowed to cooperate with the Ilon car. The AGV is allowed to cooperate with the Ilon car on the green path, where the AGV carries no load.

The visual servo is used in different experiments, where neither the distance to the scene nor the position or the heading of the Turtle are specified. The camera tilt is arbitrary. An automatic start up procedure calculates actual perspective and determines Turtle position and heading. A special reference triangle is used. The start up procedure localizes the robot and the reference symbol in the image. Image 3 shows a photograph of the output of an edge-detection. This is the information to be interpreted by the further steps of the image analysis, and the program is capable to successfully distinguish between the marking symbols and the other edges. The spatial dimensions in the experiment are given in the map of the floor in Figure 3.1. The positions of the objects are represented in a floor coordinate system relative to the reference triangle. The length unit is meter. The part of the floor seen by the camera in the actual experiment is marked in the map. The map is generated by the system as follows. The size of the reference symbol is known so the distance from the camera can be calculated from the size of the reference symbol in the image. The camera tilt is determined from the shape of the reference symbol in the image. The angle between the optical axis of the camera and floor is 32 degrees. The camera position is in the coordinate system (-0.45, -2.45) and the height above the floor is 1.85 m. Hence the distance between the triangle and the camera is 3.10 m. The angle between the line through the camera and the reference triangle and the floor plane is 36 degrees. The estimated values obtained in the actual experiment were 3.15 m and 34 degrees respectively.

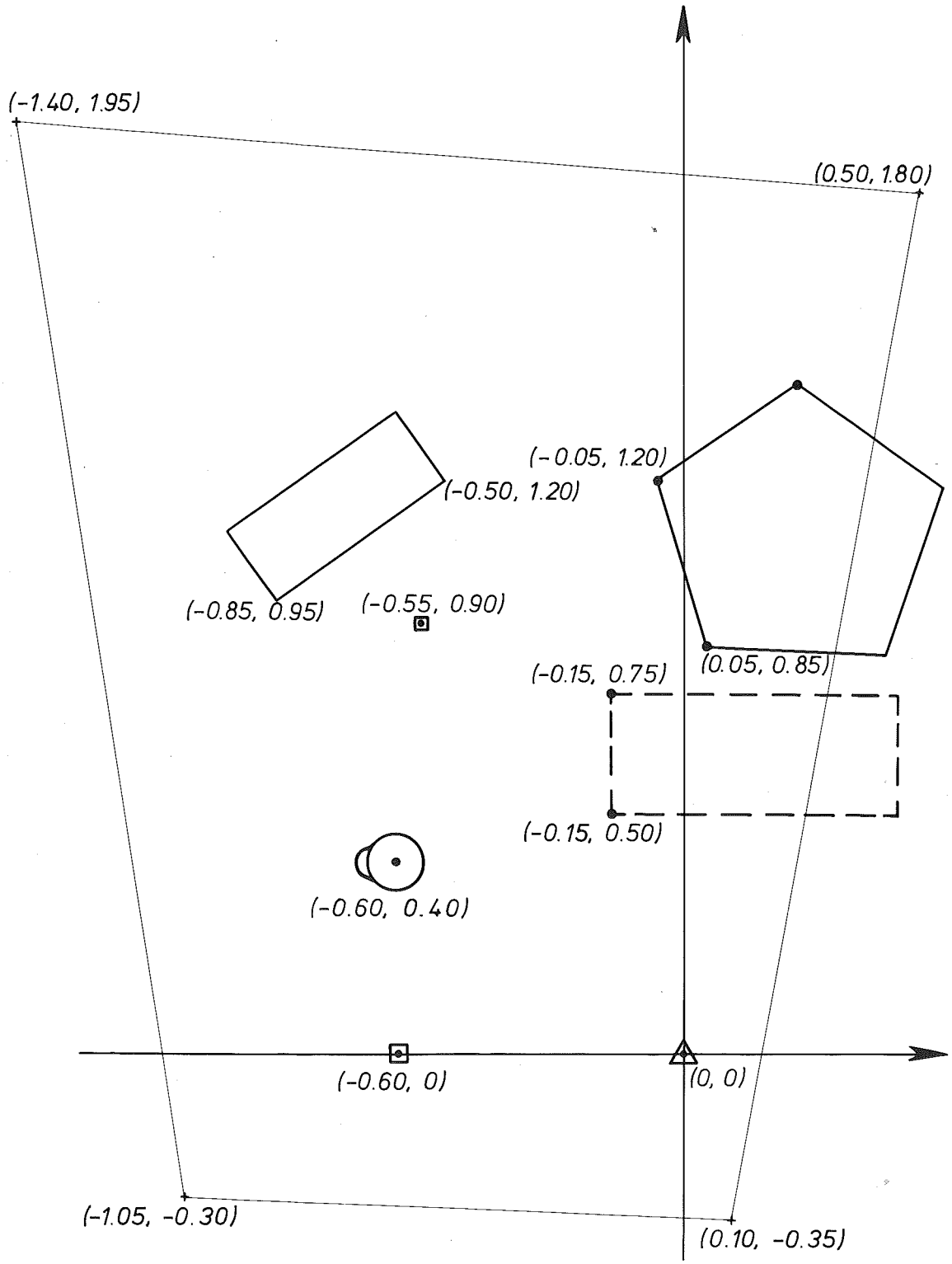


Figure 3.1 **A map of the floor showing the spatial dimensions of the experiment. The map is generated by the system. The positions of the objects are given in a coordinate system relative to the reference triangle.**

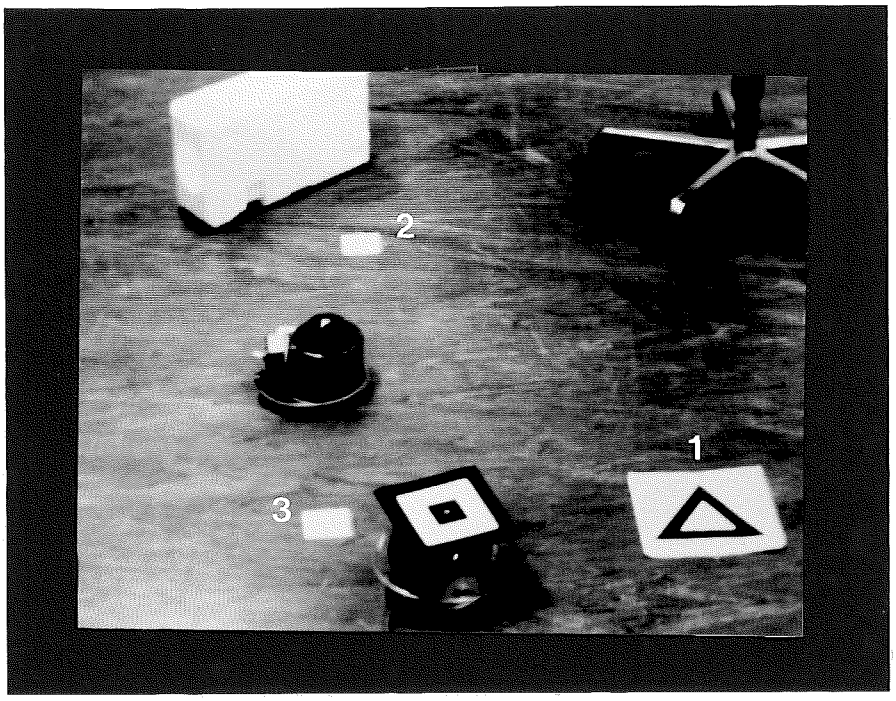


Figure 3.2 A photograph of the monitor screen showing the scene as seen by the camera at the start of the experiment.

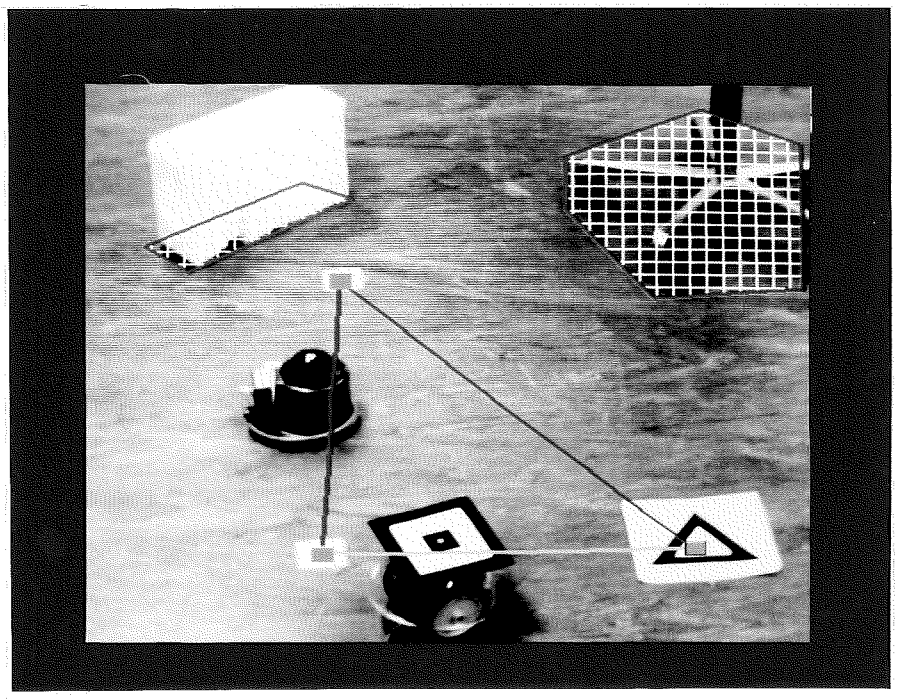


Figure 3.3 The monitor screen after the completion of the programming. The man-robot interface has been used. The result of the programming is presented in graphics which is overlayed the image of the scene.

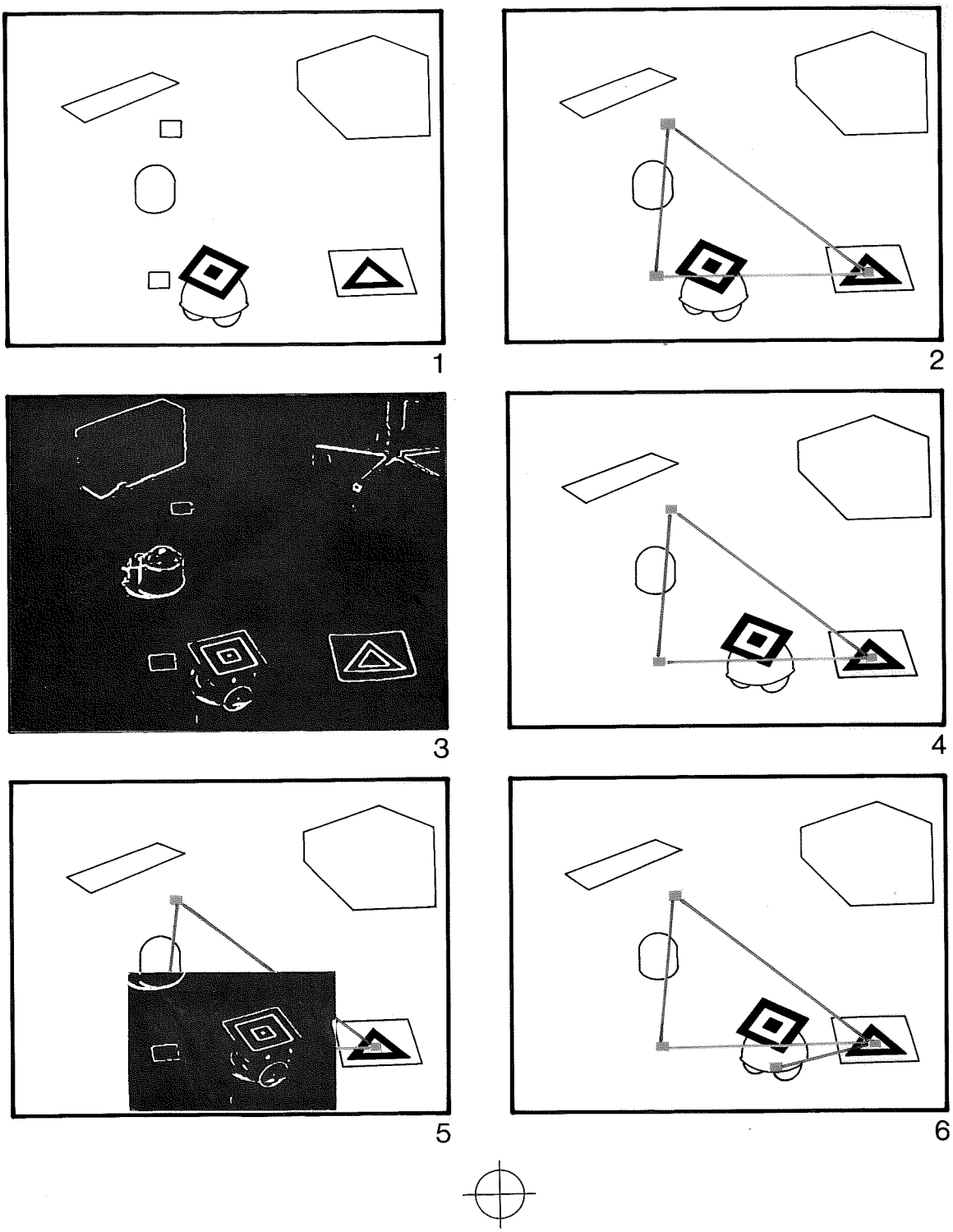


Figure 3.4 Images 1-6

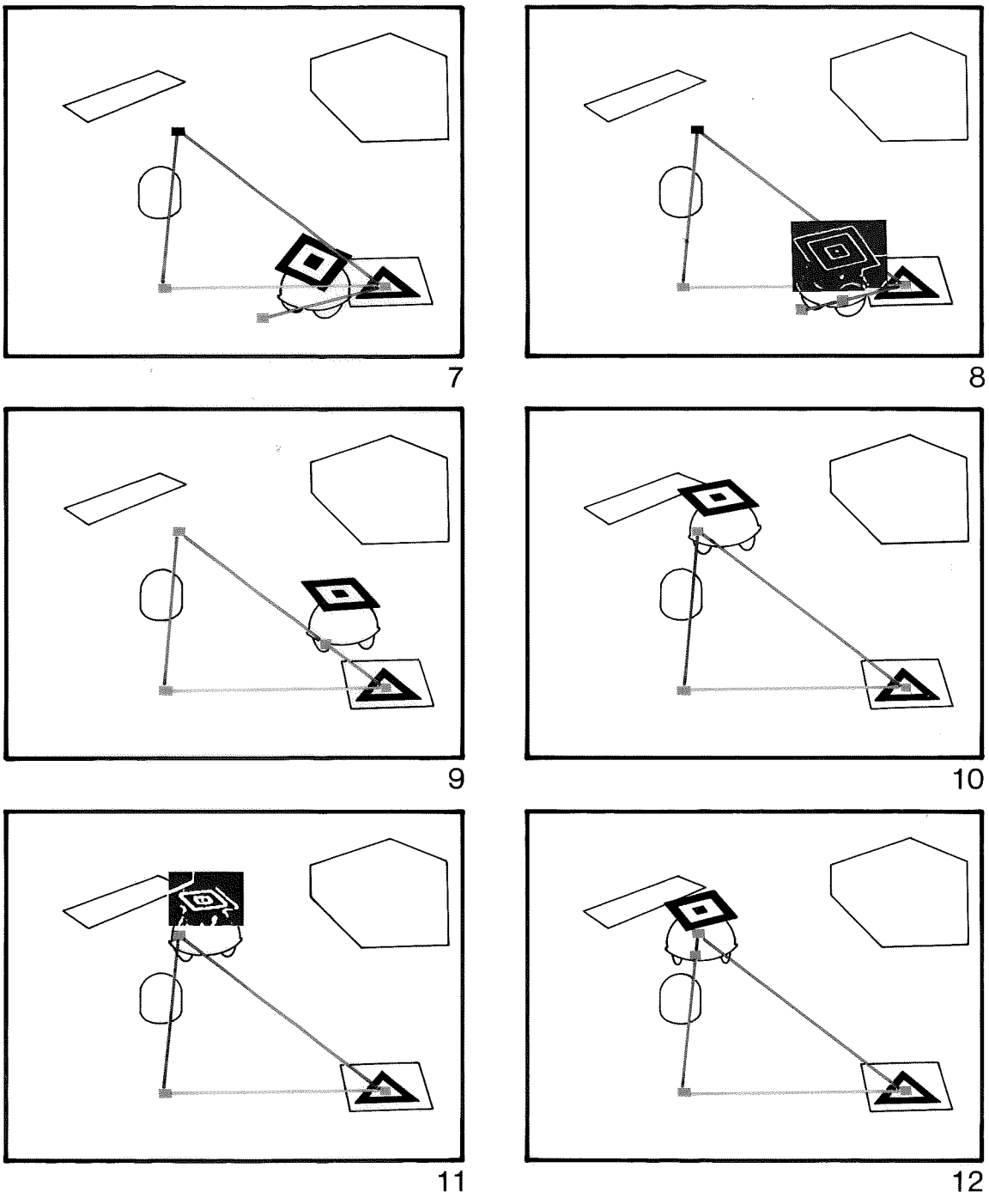


Figure 3.4 Images 7-12

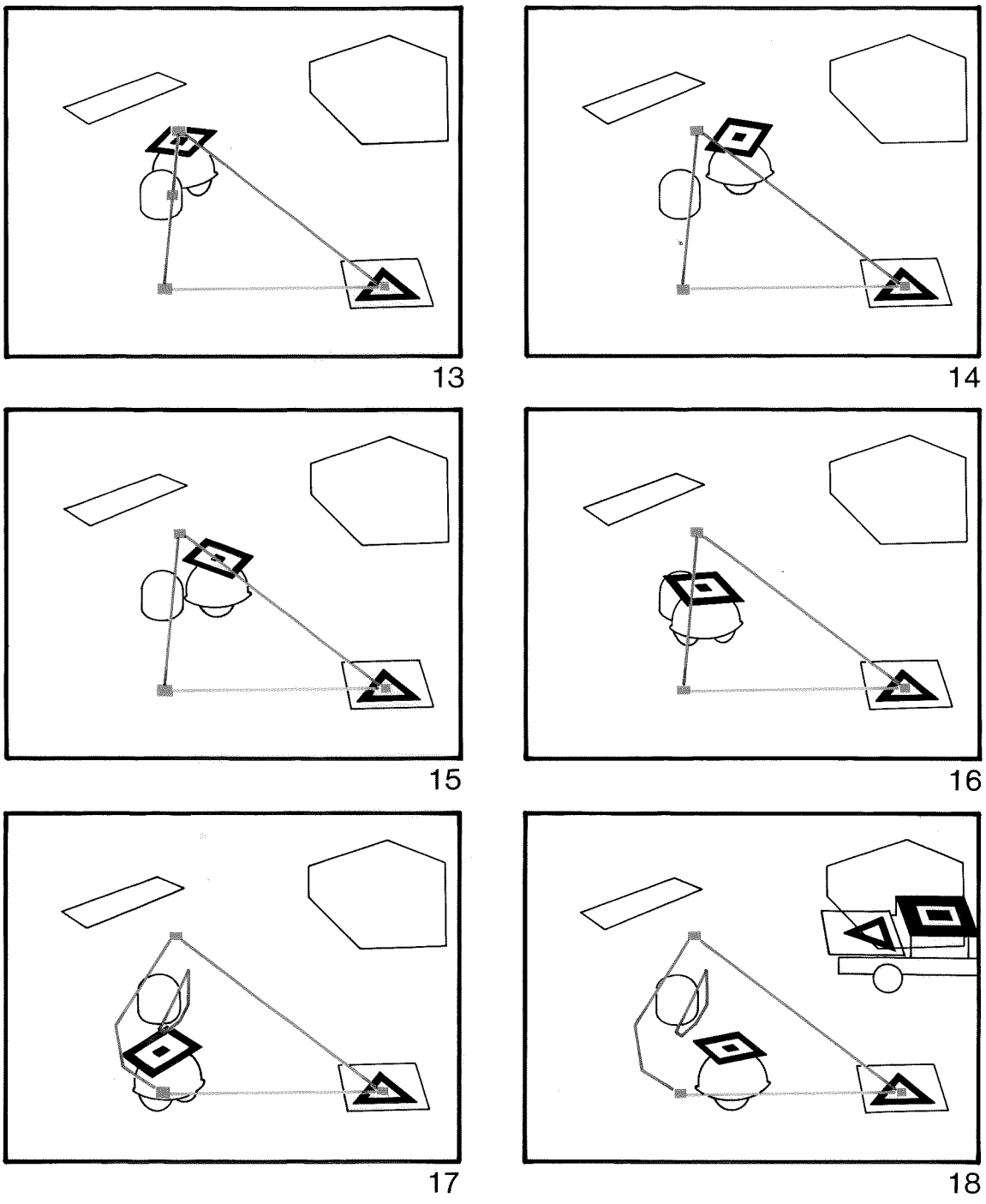


Figure 3.4 Images 13-18

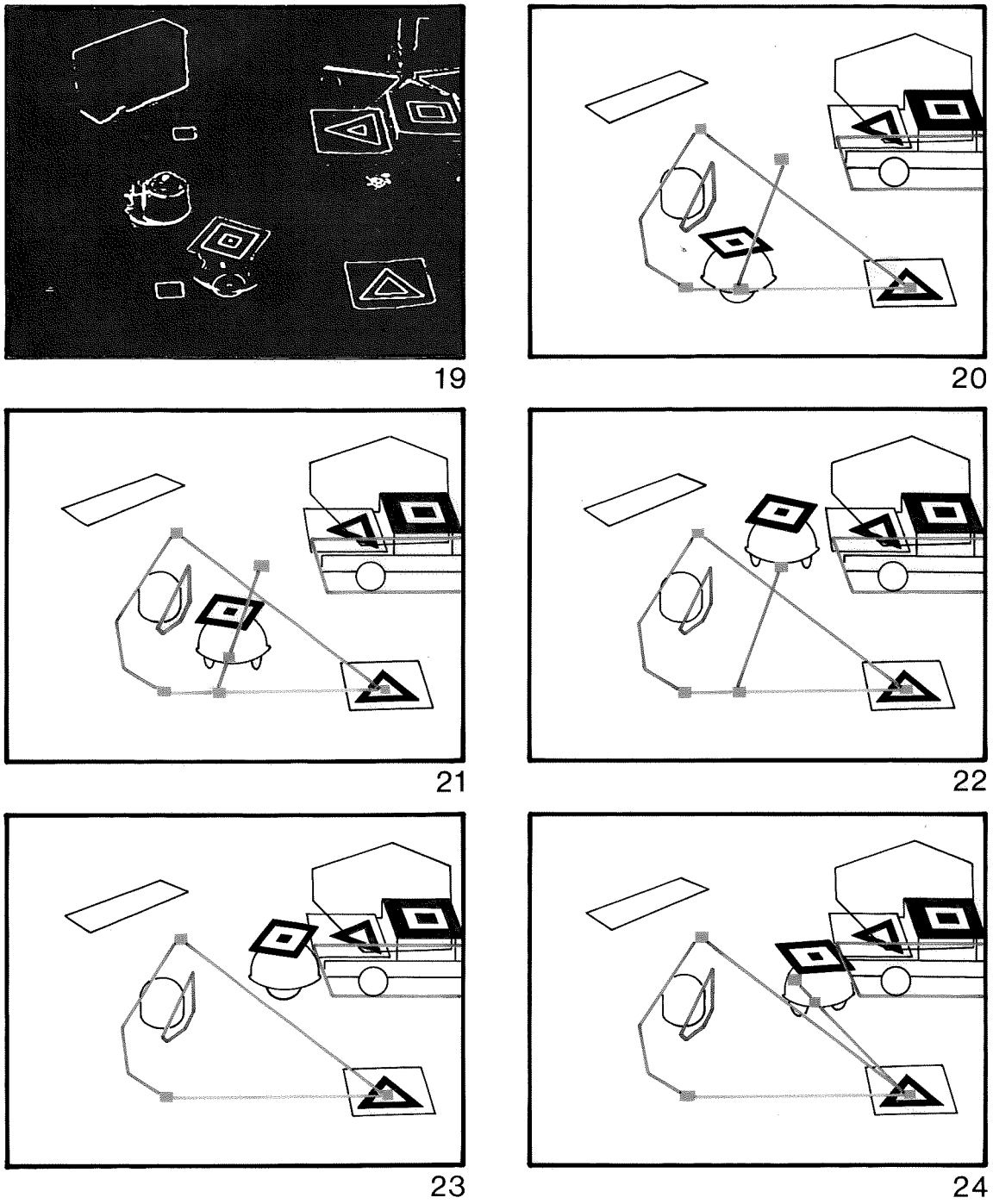


Figure 3.4 Images 19-24

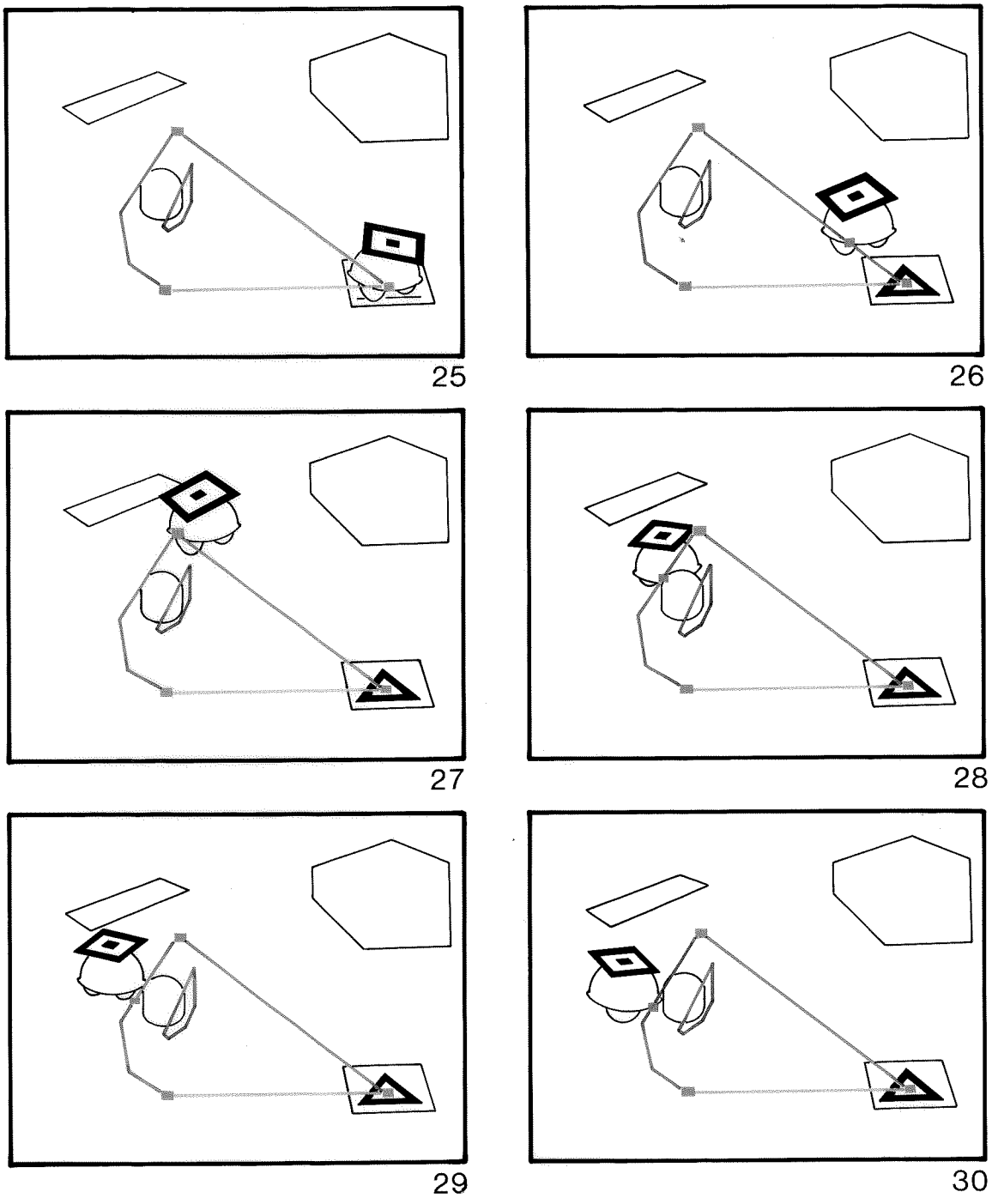


Figure 3.4 Images 25-30

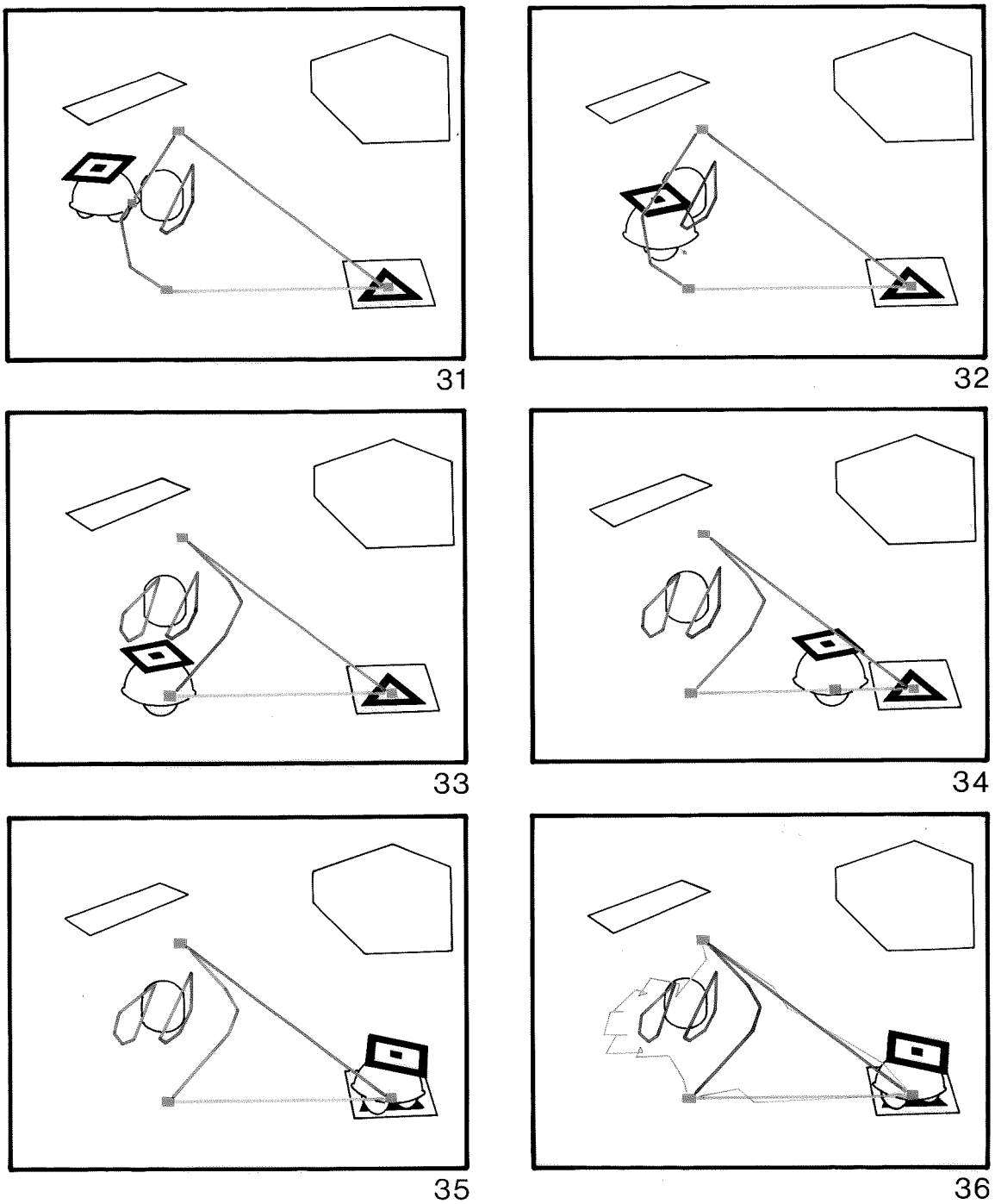


Figure 3.4 Images 31-36

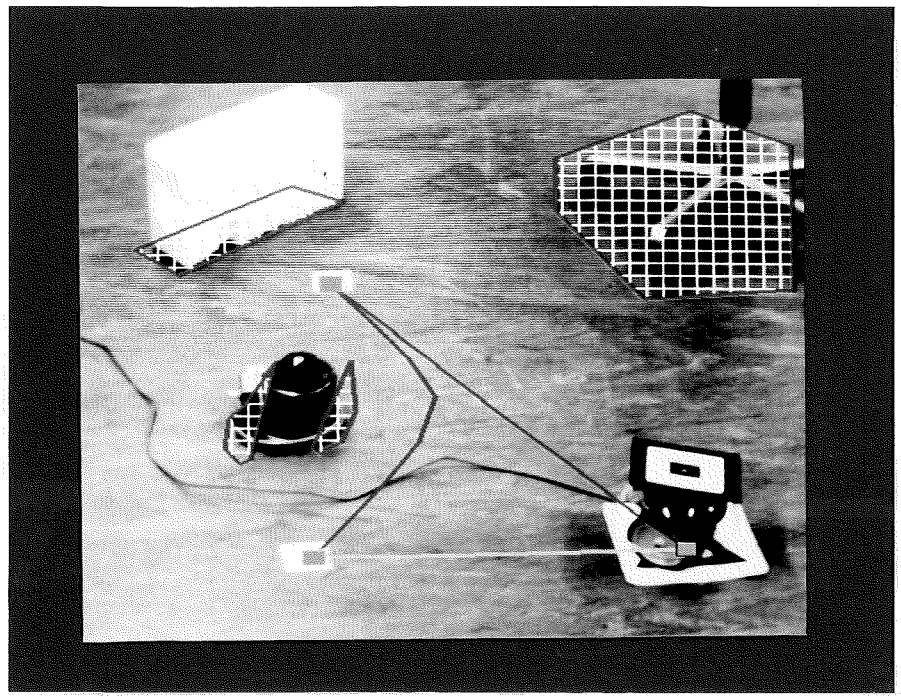


Figure 3.5 The monitor screen after two rounds of the workcycle. The description of the workspace has been automatically updated.

The image sequence continues with a presentation of the start up motion. In image 4 the AGV moves forward a short distance. Image 5 is image 4 overlayed with a photograph of the output of the edge detector. The subimage is chosen large enough to enclose the AGV whatever direction it has moved. The robot is localized and the robot heading is determined from actual and previous position. The path finding algorithm is used to calculate a way to the Station 1. The path is presented on the monitor screen as in image 6. The AGV is then controlled along the path to the first station. In image 7 the first segment of the path is done. The actual set point for the position is presented on the screen. Only a subimage around the predicted position is used in the position measuring by image interpretation during the path following. Image 8 is the same as image 7 but shows the output from the edge detector within the actual subimage.

Performing a workcycle

In image 9 the AGV has visited the Station 1, and started the workcycle by visits to the stations. The initial path from the starting position to Station 1 is not presented on the screen any more. In image 10 the AGV has arrived to Station 2. Image 11 shows the output from the edge detector within the actual subimage. Image 8 and image 11 show the change in size and shape of the marking symbol of the Turtle. In image 12 the AGV is on its way to Station 3.

A collision

On the way to Station 3 the robot collides with the unknown obstacle (image 13). The touch sensors indicate the collision. The robot starts obstacle avoidance by touch. The selection of search direction finds from the obstacle map no preference direction, and hence the default of left turn is chosen as shown in image 14. The second collision appears in image 15. The robot has moved forward according to its heading from the Station 2 to Station 3. If it had moved backward then the search direction would have been changed to the opposite. The contour following by touch continues, as in image 16, until the original path is crossed on the other side of the obstacle. Then the robot continues to its goal, Station 3. The positions of collision of the robot during the contour following are determined from the images and stored. The obstacle description is calculated, and is included in the obstacle map, and the path is replanned. In image 17 the AGV has arrived to Station 3. The updated workspace is presented on the screen. Note

that the planned path actually means a collision with the uninvestigated left side of the now partly known obstacle.

Robot cooperation

On the way back to the first station the Ilon car, shown in image 18, signals ready to cooperate. The AGV is on the green path and hence the event is treated immediately. Image 19 is a photograph of the output from the edge detector. The marking symbols of the Ilon car are localized in the image, and the area on the floor occupied by the Ilon car is temporarily inserted in the obstacle map. The AGV then avoids unnecessary collisions with the Ilon car. The path finding algorithm is then used to calculate a way to a point 10 cm in front of the rendez-vous point at the front of the Ilon car. The result is seen in image 20. In image 21 the AGV has moved the first segment and in image 22 the AGV is at the end of the temporary path. The temporary path is now deleted. From the end of the temporary path the AGV moves straight to the Ilon car until they touch in a symbolic cooperation (image 23). Then the robot goes back 10 cm. From that position a path is planned to the Station 1, and the robot is on its way in image 24. In image 25 the AGV has arrived to Station 1. The temporary path has been deleted. Furthermore the Ilon car signaled when it left, and the temporary obstacle was then deleted. The AGV continues with the second round trip of the workcycle in images 26 and 27.

A new collision

In image 28 the AGV has once again collided with the unknown obstacle. This time the algorithm for selection of search direction discovers an obstacle to the left of the path. Hence the contour following starts by turning to the right as seen in images 29 and 30. The contour following by touch continues in image 30 and 31. The collision positions are determined and stored in the same way as last time. The path is crossed in image 32, and the AGV then continues to Station 3. In image 33 the AGV has arrived to Station 3. The obstacle map has been updated and the cycle path has once again been modified. The AGV continues to Station 1 in image 34, and has in image 35 completed two round trips of the workcycle. A photograph of the monitor screen corresponding to image 35 is given in Figure 3.5. The actual motion during the second cycle is presented in image 36.

The start-up and the completion of the two round trips have been performed completely automatically by the robot without any human guidance or help. A third cycle can now be completed without collisions.

3.2 Formal Description of the Scenario

This and the next two sections describe the structure of the experiment. A human operator is assumed to define the workcycle. The operator is not assumed to be active or even present during the repeated cycle. We assume here that the description of the tasks is given. The programming i.e. how to enter this description using the man-robot interface is treated in Section 3.5. This section treats the objects, Section 3.3 treats the planning, and Section 3.4 the actions.

The concept of event points is introduced. An event point is a point on the floor which the Turtle must visit to handle an event. There are a number of available actions for the event handling. The concept of path attributes is introduced to describe what actions are allowed. The attributes are associated with each path between two stations. The values of the path attributes on a path determine whether or not an event may be treated by the Turtle on that path. The reason is as follows. It is not always suitable to leave the workcycle path. Typically the robot is not allowed to do excursions when carrying a heavy load. It has to wait until the load is delivered.

Abstract description of the workspace

The scenario will now be formalized. The notion of workspace is adopted (Brady et al, 1982). The extent of the workspace is here all points the robot can reach and the camera can see. The elements in the workspace are easily abstracted to geometrical objects in the floor plane. The elements are obstacles, stations, event points, and paths. The geometrical objects are points, curves, and areas within closed curves. The curves are restricted to be polygons in the actual implementation. The elements are listed as follows.

Obstacle:

An obstacle is an area on the floor within a closed curve. It is a forbidden area for the robot. The obstacles entered by the operator are assumed to be stationary. The event handling algorithms may add both stationary and temporary obstacles.

Obstacle map:

The set of all known obstacles is called the obstacle map.

Station:

Point on the floor. The stations are assumed to be fixed in position. The set of stations are ordered in sequence. The robot should visit the stations in the so defined order.

Event point:

Point on the floor, which the robot has to visit to handle an event.

Path:

A path is a curve (polygon) between two points on the floor. There are two types of paths. The path between two stations is called a station-to-station path. The other type is temporary paths. A temporary path is the path to or from an event point. The visual servo checks that the paths avoid the obstacles in the obstacle map. The paths are here restricted to polygons with path segments of length at most 15 cm.

Workcycle:

If the paths between the stations form a closed route, which the robot should run repeatedly, then the route is called a workcycle.

Path attribute:

The path attributes are cooperate, verify, and explore, and the value of an attribute is either true or false. The attributes are associated with the station-to-station paths. They are used to control the actions to be taken in the event handling as described in Section 3.4.

The description of the work space is manipulated both by the operator and the event handling algorithms.

3.3 Motion Planning

This section is devoted to path finding. The path finding problem is simple to state: Given an obstacle map, a start point, and a terminal point, then find a robot path that avoids the obstacles. The general and complete solution seems to be very complex. Different path finding algorithms have been studied in the literature, see e.g. Brady et al (1982) which includes further references.

Two approaches to robot programming have been identified in Brady et al (1982).

1. Explicit programming: The user specifies all the motions to accomplish a desired path.
2. Model-based programming or task level programming: The user specifies geometric models and descriptions of tasks in terms of these models. The detailed motions are derived automatically from these specifications.

Assume that we want to move the Turtle from A to B. Explicit programming can be done either by giving an explicit list of coordinates or by guiding the robot manually and store the path. In model-based programming a typical command is: move from A to B. The movement may be nontrivial and may need a motion planning or path finding algorithm. Languages for task level programming have been reported (Brady et al, 1982).

Path finding

A very simple approach is taken to the path finding problem. This is legitimised by the possibility to send an alarm to the operator. A further reason for a simple algorithm is that the motion disturbances can cause numerous collisions on a path of minimal length which worms between the obstacles. The input to the path finding algorithm is the obstacle map, the starting point and the terminal point of the path. If the calculation is successful then the output is a path which avoids the obstacles. Since the visual servo controls the robot by image feedback, we have the restriction that the robot must stay in the area which can be supervised by the camera. The restriction is described by introducing the mapping of the image boundaries onto the floor as obstacles in the obstacle map. Further details of the path finding algorithm are given in Appendix B.

3.4 Event Handling

Normally the Turtle repeats a workcycle. The visual servo is provided with some flexibility to handle asynchronous events when working unsupervised. They result in signals sent to the servo. Three different events may appear. The Turtle can interact with another robot. The other robot is assumed to signal when it comes or leaves. Secondly, objects may unexpectedly appear in the work space. Humans may e.g. be entering a risk zone. An external motion detector signals if something is moving into the scene. Thirdly, the Turtle has touch sensors. The touch sensors are used to detect collisions with obstacles and to explore the scene.

The presentation will proceed by first describing the control structure, which resembles an operating system of a computer. It is here called a motion kernel. The actions are then treated.

The motion kernel

The motion kernel controls the robot actions. Its purpose is to give the robot a path to follow. The information available to the kernel is the geometric description of the workspace, the state of the robot, and the event signals. The kernel introduces the event points and the temporary paths, and may introduce obstacles in the obstacle map. It replans the station-to-station paths according to the updated obstacle map. The stations, the path attributes, and the obstacles defined by the operator are never changed.

The concept of event list will now be introduced. An event signal from the other robot or from the moving-object detector requires a check of the path attribute values on the present path. If the event is not handled immediately then the event is put in a list of events yet untreated. For each event in the list the type of event and the corresponding event point are stored. The event list is scanned by the motion kernel each time the robot is at a station. It is checked if the path attribute values for the next station-to-station path allows handling of any event in the list. An event will not be treated if there is no path in the workcycle with attribute values that allows handling it.

An outline of the motion kernel with possible events, path attributes, event points, and actions is described as follows.

```
MotionKernel is

Background job:
  Action: PerformWorkcycle

Collision event (First priority):
  Action: ObstacleAvoidance

Object detection event (Second priority):
  Path attributes: verify and explore
  Event point: detected object position
  Action: if verify and explore then
            ExploreAction
          else if verify then
            VerifyAction

Robot cooperation event (Third priority):
  Path attribute: cooperate
  Event point: 10 cm in front of the other robot
  Action: if cooperate then
            RobotCooperation
```

There is an alarm possibility at all levels if an action cannot be completed. If it is possible to continue the robot only notifies the operator. Otherwise it calls for help and waits until the operator intervenes.

The actions of the motion kernel are basically composed from the following subactions: path following with image feedback (PathFollowing), contour following with the help of touch-sensors (ContourFollowingByTouch), and path finding (PathFinding). The algorithms for the path following and the contour following are given in Appendix B. Path finding is discussed in Section 3.3 and Appendix B.

PerformWorkcycle

The normal mode of operation is PathFollowing on the paths of the workcycle.

ObstacleAvoidance

A collision must be treated immediately. Collision with an unknown obstacle is the normal case. The robot may also collide with a known obstacle when it due to motion disturbances deviates from the planned path. The obstacle avoidance works as follows.

```
ObstacleAvoidance is
  if OldObstacle then
    ReturnToPath
  else
    begin
    SelectionOfSearchDirection
    ContourFollowingByTouch
    end
```

The test for the case of old obstacle or not is based on the measured distance to the obstacles in the obstacle map. The Turtle position is determined from the image. If the Turtle position is within the distance of one Turtle diameter to a known obstacle in the obstacle map, it is assumed that it is a collision with the known obstacle. The Turtle then moves away from the obstacle to return the shortest way to its path.

In a collision with a new obstacle the Turtle has to decide on starting the contour following to the left or to the right. The decision is not crucial for the ability to find a path, because if one search direction is unsuccessful then the other search direction is automatically tried. However a good guess speeds up the contour search. The guess is based on the obstacle map, the planned path, and the state of robot motion, see Appendix B. An obstacle contour is obtained from the contour following algorithm and the obstacle map is updated. The path finding algorithm is called to replan the paths of the workcycle to avoid the new obstacle.

Obstacle avoidance is a planning problem with incomplete information, and there are dual aspects which may be considered in a further development. The investigations by contour following are time consuming. There is therefore a balance between investigation and heading for the goal. The judgement should be on how valuable the investigation may become for future time savings or for safety reasons. If the collision has occurred on a station-to-station path the obstacle knowledge is valuable, since the Turtle will return. Another aspect is that the operator may wish the robot to follow the path as close as possible. A collision on a temporary path is perhaps not as interesting. The robot could just try to find a free way and replan its continued motion. There are also compromises in the selection of search direction. If the collision occurs on any side of the robot, it should prefer to turn the other way unless there are known obstacles close in the obstacle map. Close to the image boundaries the robot should prefer to move towards the center of the image.

VerifyAction and ExploreAction

An external moving-object detector can indicate a new object moving into the scene. The moving-object detector also sends the detected object position, which is the event point for this event. There are two possible investigation actions.

VerifyAction: The path finding algorithm is called to plan a path to the event point, and the Turtle goes there. The Turtle verifies the presence of an object by the use of its touch sensors. The information is used e.g. to distinguish between objects and shadows. The path finding algorithm is called again to plan a path to the next station, and the Turtle goes there.

ExploreAction: As in the verify action the Turtle goes to the detected object position. If an object is sensed, then the contour following algorithm is called. The Turtle will as a result of this encircle the object, and an obstacle contour is obtained from the contour following algorithm. The obstacle map is updated and the path finding algorithm is called if any path in the workspace needs replanning. The path finding algorithm is called again to plan a path to the next station, and the Turtle goes there.

RobotCooperation

The following sequence of actions is performed in the case of robot cooperation. The Ilon car (see Chapter 2) is used. The size and shape of the Ilon car is known. Both the position and the orientation of the Ilon car is determined from the image, and the area on the floor occupied by the car is calculated. The event point 10 cm in front of the Ilon car is also calculated. The obstacle map is temporarily updated with the Ilon car as an obstacle. The path finding algorithm is called to plan a path to the event point. The Turtle goes there using the path following algorithm. The Turtle then starts a rendez-vous operation by moving straight to the Ilon car until the touch sensors verify contact. The Turtle then returns to the event point. This is a simulation of e.g. a load transfer. The path finding algorithm is called again to plan a path to the next station. The Ilon car signals when it leaves and the temporary description of the Ilon car as an obstacle is then removed from the obstacle map. The Turtle continues the workcycle.

3.5 The Man-Robot Interface

A man-robot interface has been developed to let the operator interact with the system. It was inspired by the graphics interaction facilities in recent personal computers (Kay and Goldberg, 1977; Williams, 1983). An overview of graphics techniques can be found e.g. in Newman and Sproull (1979). An important feature of the man-robot interface is the explicit use of real images of the scene. The programming is done with color graphical manipulations on these images, using a simple graphical editor for the interaction. Furthermore, the man-robot interface provides the possibility to follow the status and the internal algorithms on the screen. The man-robot interface is used to describe the workspace i.e. to enter stations, obstacles, paths, and path attributes (see Section 3.2). Hence the facilities of this interface is concentrated on motion. Other important aspects such as gripping and manipulation are not treated.

Correspondence between image and workspace

The correspondence between the image and the workspace is simple. The mapping between the floor plane and the image is one to one. The description of the workspace consist of geometrical objects on the floor. These points and curves have a unique representation in the image. Conversely a point or a curve in the image has a unique representation in the workspace. The geometric description of the workspace may thus be entered by pointing in the image.

Interactive tools

The terminal, the mouse, the joysticks and the monitor screen described in Chapter 2 (see Figure 2.1 and 2.2) are used for the interaction. There is a manual mode where the robot is controlled using the joysticks. The mouse is used for pointing in the image on the monitor screen.

The graphical editor

The visual servo uses gray scale images of the scene. These images are presented on the monitor screen. The output of the graphical editor is overlayed on these gray level images. Figures 3.3 and 3.5 are typical examples. The function of the editor is such that if a graphical object e.g. a line is moved then the image hidden by the line is restored. The commands to the editor are selected

from menus on the terminal.

The editor can handle points and curves. Two types of curves are possible: polygons and cubic B-splines (Newman and Sproull, 1979, Section 21.4). The shape of a polygon is defined from its vertices. The shape of a cubic B-spline is defined from control points. An illustration is seen in Figure 3.6. The technique with cubic B-splines is well established. The control points have local support. This means that if a control point is moved then the curve is changed only in the neighborhood of that point. It also satisfies the requirements of shape stability in editing, which means that the shape of a curve is not drastically changed if a control point is moved slightly. An overview of the requirements on shape representation is given in Newman and Sproull (1979, Chapter 21).

The editor can be used to interactively enter, copy, move, rotate, or delete both points and curves. The shape of a curve is edited by adding, moving, or deleting the shape defining points (a polygon vertex or a control point of a cubic B-spline). Assume that we want to move a vertex of a polygon. A typical interaction sequence is then: give the command edit curve, point on the actual curve with the mouse, (the curve changes to violet to confirm the selection), give the command move point, point on the actual point, and point on the desired position of the point. The curve is redrawn and the color returns to the original.

Path programming

When a new station is entered a path avoiding the obstacles is calculated by default, using the path finding algorithm, and presented to the operator. The operator may conveniently modify the calculated path or program a new path as follows. The shape of a curve is entered using the graphical editor. The curves obtained in this way are used only as a tool in the programming, and they are transformed to Turtle paths. This is done by sampling the curve to a polygon on the floor with no segment longer than 15 cm as illustrated in Figure 3.6. The program of course checks that the obtained path avoids the obstacles. The result is a smooth polygon path. The extension to other types of shape representations in combination with other types of dynamical behaviour for different robots seems to be an interesting research field. The desired shape of a path is defined interactively and the man-robot interface calculates a path both possible for the robot to follow and if not exactly the same then close to the defined shape.

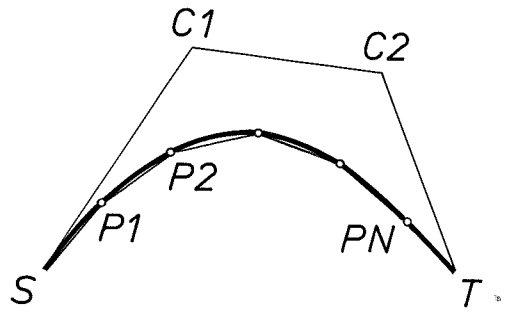


Figure 3.6 The idea of path programming with the help of shape defining splines. A change of the control points for shape, C1 and C2, causes a change in the shape of the continuous curve. This curve is then sampled to give a polygon P1, P2, ..., PN. S is the starting point of the path and T is the terminal point.

How to enter the elements

The commands are used to determine whether the graphical input is obstacles, stations or paths. The coordinates in the image are transformed to points on the floor. The programming elements i.e. obstacles, stations, paths and path attributes are entered in the following way.

Obstacle:

Entered as a closed polygon. The vertices are pointed out directly in the image.

Presentation: Red polygon filled with a white grid net.

Station:

The point on the floor representing a station may be entered in three ways.

1. Point in the image with the mouse.
2. Control the robot to the position of a station using the joysticks. Give a command which tells that this is a station. The robot position is determined from the image.
3. Tell the robot that a certain feature in the image is a station. The only features accepted as a station definition are the specified marking symbols. Ideal would be e.g. "The table is next station". This is not possible in the actual implementation.

Presentation: Blue dot.

Path:

The graphical editor and the mouse is used to define a number of points. The points are by command interpreted as a polygon or (default) a cubic B-spline. Both the spline and the sampled Turtle path are presented during the editing, and thereafter only the Turtle path is presented.

Presentation: Polygon in color according to the values of the path attributes.

Path attributes:

Entered by placing the cursor on the path and then by commands which give the desired path attribute values. A color code is used to indicate the different combinations for the attribute values.

Presentation of status

There are three levels of presentation on the monitor screen during operation. Firstly, the monitor can just display the working scene as seen by the camera. Compare Figure 3.2. Secondly, the graphical description of the workspace can be overlayed the image. Further, the predicted position is marked with a blue dot. Compare Figure 3.4. Thirdly, the result of the intermediate steps of the image processing can be added to the presentation. The image processing is described in Chapters 4-6. The actual processed part of the image is marked, the output of the edge detector is displayed etc. Compare images 3, 5, and 8 of Figure 3.4.

3.6 The Implementation

Some implementation experiences will be reported. The goal of the implementation is to get a program which results in the desired robot performance. This section starts with a presentation of the overall structure of the program. Then the organization of the data base is indicated, and finally some comments on the development are given.

The program

The environment for software development described in Chapter 2 has been used to implement the visual servo. The code consists of 13987 lines of Pascal representing an amount of 387 kbytes of source code. No special attempts have been made to obtain compact code. The executable code size is 146 kbytes. The source code is distributed in the following way.

Support packages	10 %
Turtle control	27 %
Image interpretation	21 %
Man-robot interface	42 %

The support packages include the packages mentioned in Section 2.3, and furthermore a package for handling of lists and a package for handling of splines. The image interpretation code size includes the start-up estimations.

The database

The algorithms are implemented as procedures working on a database. The database contains a description of the workspace, the state of robot control, the image data, and the man-robot interface data. The Pascal program uses doubly linked lists to handle the problem of a priori unknown number of stations, number of polygon vertices (both in paths and obstacles), number of curves found in the image processing etc.

The workspace of the robot is described in Section 3.2. The obstacle map is represented by two equivalent representations, which are updated in parallel. The vertex points of the polygons representing the obstacles are stored in a list. The obstacle map is also represented by a 512 x 512 binary matrix, where the areas in the image that are occupied by obstacles are labeled. The list representation of the obstacle map is used during interaction, and the matrix representation in the path finding algorithm. The robot paths are represented by two lists. One list is used for the vertices of the sparse polygon defining the shape of the path. The other list is used for the polygon vertices of the sampled path. Each node of both lists contains the position coordinates, the path attribute and a sequence number telling which station-to-station path it refers to. The actual path is given by the motion kernel. It is either a copy of a list representing a station-to-station path, a

temporary list from path finding, or a list corresponding to the robot search motion. The list representing the actual path is the input to the algorithm for the path following.

The state of the robot control is represented by global variables. The robot position and heading, the actual motion mode (forward, backward, left-turn or right-turn), the predicted position and the actual path attributes are stored and updated. Events which are not yet treated are stored in an event list.

The images are represented as a data hierarchy. Different representations are computed from each sampled graylevel image. These are the graylevel image itself, a binary image from the edge-detector, a list of all objects, a list of triangles and squares, and a list of recognised marking symbols. The algorithms are given in Chapters 4-6.

The man-robot interface uses the representation of the workspace. The inputs from the joysticks and the mouse are represented globally. When the graphical editor is used to edit a polygon then the polygon is removed from its list in the motion space representation and inserted into a local list. The editor achieves greater speed by only working on a local list. Further the image data overlayed by graphics is saved in a list, and when the graphics is changed then the hidden image data is restored. The color look-up table (see Chapter 2) is used to combine color graphics with black and white images.

The development

The visual servo has provided an experimental platform. The programs have existed in many different versions and have been modified in different directions over time. The program modifications consist of mainly two steps: An algorithm should be programmed and then the resulting behavior should be tested. The notation in Pascal has been a help during the work. Pascal provides data structures and recursion. The type checkings made by the compiler prevented many errors. The code modifications have been streamlined using command procedures. Pascal is, however, batch oriented and not interactive. An environment for interactive program development would have been helpful to test new ideas on-line when some interesting behavior or error occured during an experiment. Here the experiment has to be repeated from the beginning, since the

status of the program is lost after a code modification. The second step in the modification of the program is the testing of the resulting behavior. The convenience of these tests mainly depends on the time needed to perform an experiment, which in turn depend on the time to execute the code. It has been a major effort in the program development to test the resulting motion. It is difficult and time consuming to design real world experiments to test the function in the possible cases of Turtle action.

3.7 Conclusions

When approaching the problem it became clear that several concepts developed in other areas fit well into the robot control problem discussed in this chapter. The event handling can be organized using ideas from real time programming. It also has a flavor of expert control (Åström and Anton, 1984), where ideas of planning and replanning are used. The tasks can be defined to the robot using ideas of man-machine interaction from modern personal computers. These aspects have been illustrated here.

The Sections 3.2-3.4 illustrated how an industrially useful behavior could be obtained using available technological concepts. Simple versions of path following, contour following by touch, and path finding were used. Different combinations of these operations gave the robot the flexibility to avoid obstacles, to cooperate, and to investigate other objects. A number of concepts were introduced. These formalize the workspace and the objects in it. The collected knowledge of the workspace is simply a geometric representation as described in Section 3.2. The event-handling described in Section 3.4 performs the necessary planning and replanning.

The color graphics manipulation of real images of the robot working scene simplifies robot programming. The graphics is entered simply by pointing in the image. The main idea is that the man-machine interface automatically translates the graphics to process characteristics. The man-robot interface in Section 3.5 illustrates these ideas. Here e.g. curves in the image were translated to paths on floor. The method relies on the fact that a point on the floor has a unique representation in the image and vice versa. The immediate application could be

the control of automated guided vehicles (AGVs), where the expenses of magnetic trails in the floor could be saved. Further the paths are simple to change by pointing in the image. This is drastically different to todays system where the paths are essentially fixed. Flexible storehouses, where the tasks for the AGVs can easily be adapted to different needs, can be obtained using our approach. In such a system storeshelves can be moved, paths can be changed and so on.

The concept of a map was used here to represent the obstacles in the work space. The map of the experiment in Section 3.1. had plenty of free space around the obstacles. Different types of maps can be used in other applications e.g. in the maneuvering of unmanned underwater vehicles using TV cameras. When inspecting the inside of pipelines the map can be a detailed CAD material from the construction of the pipeline system. An important application is to help a human operator not to get lost in the maze of pipes. In several cases signposts can be used to mark important points in the workspace. These signposts can be used to verify the position. The perspective invariant symbols in Chapter 5 offer a particularly simple way to design signposts.

Finally, the visual servo example gives an experimental background to the problems treated in the remainder of this thesis.

4. An Approach to Image Processing

Many approaches can be imagined when trying to mechanise the process of discovering what is present in the world, and where it is. Existing systems normally start with extraction of features or properties at different levels. At a basic level egdes, texture, color etc. can be extracted. Higher level properties like areas of connected components or position of corners may then be determined. Finally these properties are composed to recognition of objects. In the recognition stage it must be considered that an object gives different images depending on position, orientation, and illumination.

This chapter reviews one approach to the vision process (Petersson, 1983). The approach has proven to be of value in the industry, and there exist now commercial vision systems based on such an approach. It is implemented in special purpose parallel hardware and consists mainly of three parts: contour extraction, extraction of contour properties, and object identification. These steps are normally performed in sequence, but backtracking may be introduced in different ways. The contour extraction may e.g. be repeated with modified parameters if the object identification is unsuccessful. The presentation will serve as a background for the following chapters.

The notions of contour extraction are given in Section 4.1. Two levels of contour properties of different complexity are treated in Sections 4.2 and 4.3 respectively. The type of properties will then be fixed, and the possibility to describe objects based on these properties will be discussed in Section 4.4. The approach is used in the visual servo, and the specific details are given in Chapter 6.

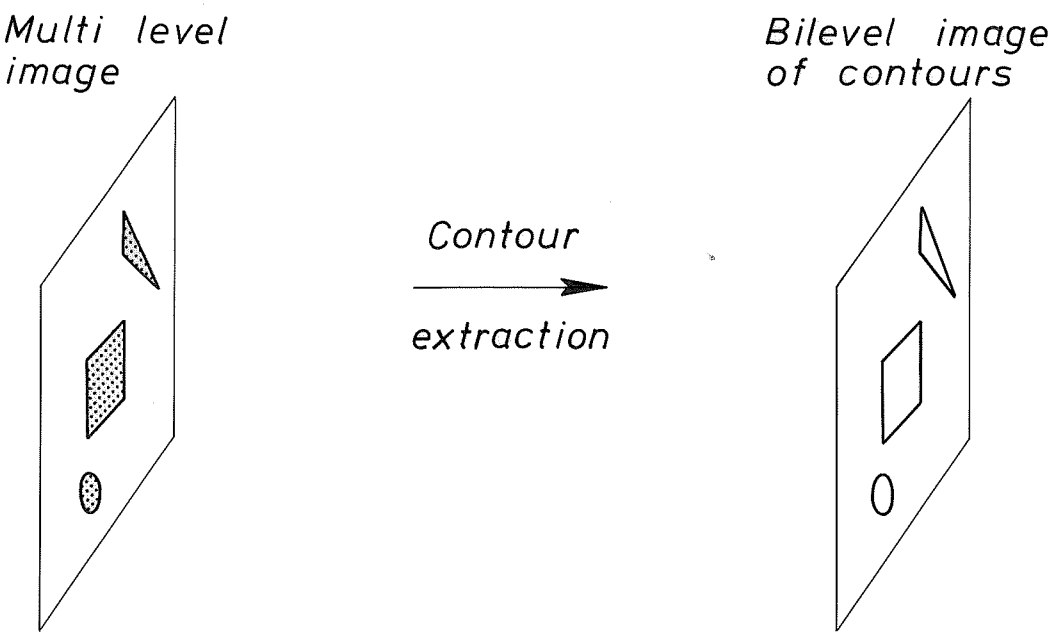


Figure 4.1 Contour extraction.

4.1 Contour Extraction

Extraction of contours is the first processing step in the approach under review. The input is a multilevel image and the output is a bilevel image representing the contours, see Figure 4.1. The contours are seldom perfect and there exist many algorithms of varying sophistication. An overview of contour extraction is given in Rosenfeld and Kak (1982, Chapter 10). The problem is basically a problem of pixel classification. A pixel is classified as a contour point or not. There are many properties which can be used for classification. The gray level is normally chosen, but other multilevel properties such as color or texture measures may be used (Rosenfeld and Kak, 1982, Chapter 10). A contour is the boundary around an area having properties different from the surrounding areas.

Here a gradient operator is applied to the image to extract the contours. The result is a multilevel image where the pixels corresponding to contours have high values compared to noncontour pixels. The classification is then done by thresholding the gradient magnitude. There exist several versions of gradient operators e.g. Roberts, Prewitt, Sobel (Rosenfeld and Kak, 1982, Section 10.2).

The notion of image parallel algorithms will be introduced. If an algorithm is based on local properties of an image and the calculation at one pixel does not depend on the calculation at other pixels, then the algorithm is image parallel. Image parallel algorithms can be implemented in special purpose parallel hardware. Algorithms that are not image parallel are called sequential.

The processing described so far has been image parallel. The refinement schemes may be parallel or sequential. Sequential algorithms (Rosenfeld and Kak, 1982, Section 10.4) can typically start in points with large gradient values. The algorithm then tries to extend or "track" contours. The image parallel algorithms are typically based on iterative refinement of the original gradient images. This approach is called relaxation (Rosenfeld and Kak, 1982, Section 10.5).

4.2 Integrated Measures

The contours in the bilevel image may be described in different ways. Integrated measures of the contours like the centroid ("of gravity"), or the perimeter of a curve, or the area inside closed curves are simple possibilities. The bilevel image is processed and the output is here a MeasureList (Figure 4.2) where each curve is represented by its position, area, and perimeter. The identification of objects is then based on this list. An example is given in Section 4.4.

4.3 Contour Attributes

The curves of the bilevel image (Figure 4.1) may be processed further to obtain a more structured description of the contours. Different shape attributes may be extracted using different techniques (Pavlidis, 1978 and 1980; Fu, 1977). The attributes used are highly dependent on the intrinsic properties of the objects. Objects where corners are good shape measures will be discussed further in this work.

Typical calculations are illustrated in Figure 4.3. The first step here is to obtain a complete description, ContourList, of the curves in the bilevel image. A curve may be represented in many different ways e.g. a list of all points, or a starting

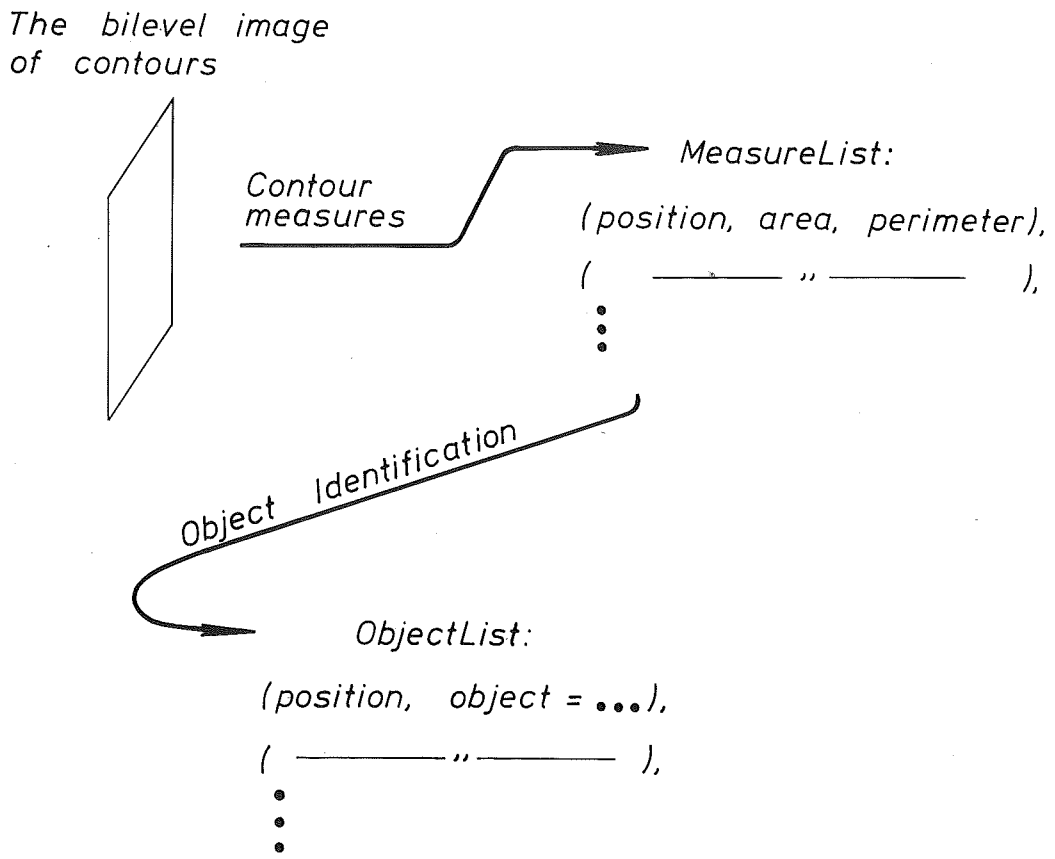


Figure 4.2 An illustration of object identification based on integrated contour measures.

point together with a chain code. The chain code gives the direction to successive points of the curve. Further examples are given in Rosenfeld and Kak (1982, Section 11.1). Attributes are extracted from the curves. The result is an AttributeList where each curve is represented by its attributes. Finally the identification of objects is performed using the AttributeList. In our work we will take the attributes to be position, area, perimeter, type of shape and position of corners. By the type of shape we mean the number of corners. A simple shape classifier is presented in Section 6.2. It can classify between a triangle, a quadrangle, or something else.

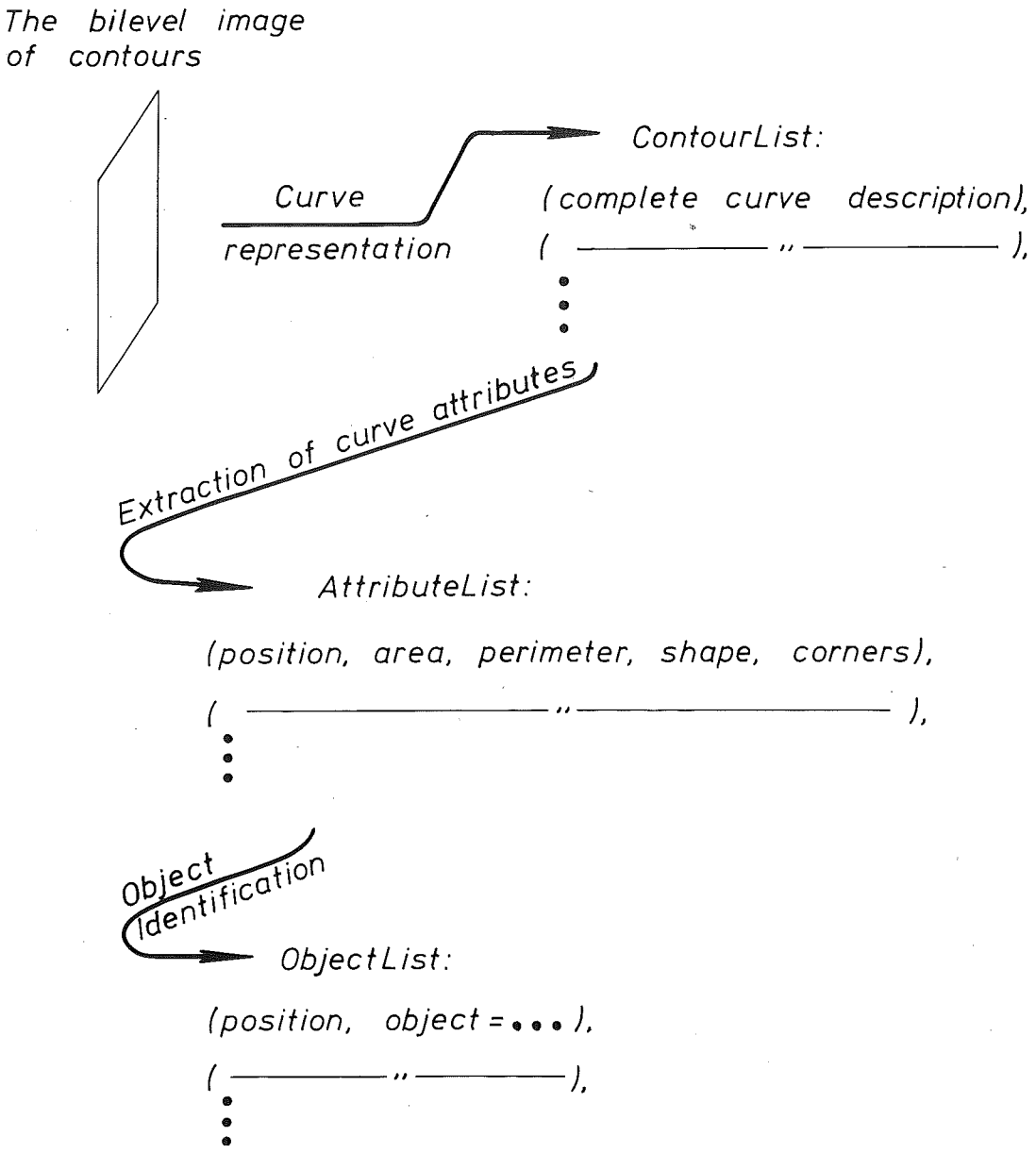


Figure 4.3 **An illustration of object identification based on localization of contour attributes.**

4.4 Object Description

The processing discussed so far has resulted in a description of the contours, available as MeasureList or AttributeList. Denote the obtained properties by a. In general an object description is a function F(a) of the extracted properties. The function F may be very complicated but takes different values for different objects. The construction of a description F must consider the domain of a. The properties $a(I(\xi))$ are functions of the image I, and the image is in turn a function of the position and orientation ξ of the object. A description F(a) is particularly simple if it is invariant to ξ.

Consider as an example images taken from a camera mounted above a conveyor belt which contains squares and other two-dimensional objects. The objects may arrive on the conveyor belt in an arbitrary position and orientation within the plane of the conveyor belt. The positioning variable ξ is then restricted to describe two-dimensional rotation and translation. The perimeter P and the area A inside closed curves, and the number of corners are independent of position and orientation. These properties are thus invariant under two-dimensional rotation and translation i.e. independent of ξ. The shape index $S = A/P^2$ is thus also an invariant (Brady and Yuille, 1983), since it is a function of invariants. The object identification is illustrated by a simple example.

Example - Use of contour properties for object identification
Assume that it is desired to distinguish a square with side b from other squares and from triangles and circles. Assume that the MeasureList shown in Figure 4.2 is available. We have then

$$a = (A,P)$$

A possible description could e.g. be

$$F(a) = (a,S)$$

where we have the values $A=b^2$, $P=4b$, and $S=1/16$. This description is independent of ξ and is easily computed from MeasureList. It can thus be used for identification. Squares of different sizes can be separated using the property A. They can be distinguished from a triangle with area b^2 by the value of P or S. The AttributeList in Figure 4.3 introduces the further possibilities to test on the number of corners and their relative position. □

The situation changes in an application where the objects move in three dimensions or where they are arbitrarily oriented in a bin. In this case the variable ξ represents three-dimensional translation and orientation. The shape and size of contours then vary. Two squares of different size are e.g. not distinguishable, since they can give exactly the same image I if the larger square is viewed in a more distant position. The extracted properties a(I) are then the same, and any description F(a) is of course the same.

Further study of descriptions F(a), where a is MeasureList or AttributeList, in the three-dimensional case is the objective of Chapter 5.

5. Perspective Invariant Marking

It is not possible to identify all types of objects in the case of three-dimensional imaging. For example two squares of different size are not distinguishable. An interesting question is if there exist classes of objects which can be described independent of their position and orientation. In this chapter we will design such objects. This means the construction of symbols and their description. The descriptions we are looking for are invariants of perspective, so that the symbols are distinguishable based on the invariants. The invariants should be functions of contour descriptions which are easy to obtain from image processing. It is desirable that the invariants are based on integrated measures, e.g. the properties of the MeasureList of Section 4.2. It is well-known that the quotient between two areas of a planar figure is invariant under parallel projection. This is independent of the shape of the areas. Extensions of this result to perspective projection will be discussed.

An algebraic description of imaging is given in Section 5.1. A number of properties are given in Section 5.2. After these preliminaries the precise problem formulation is given in Section 5.3. Useful invariants based on areas are derived in Section 5.4. Results for contour attributes are given in Section 5.5. The invariants are simplified if the approximation of parallel projection can be justified. This is exploited in Section 5.6. Some conclusions are given in Section 5.7.

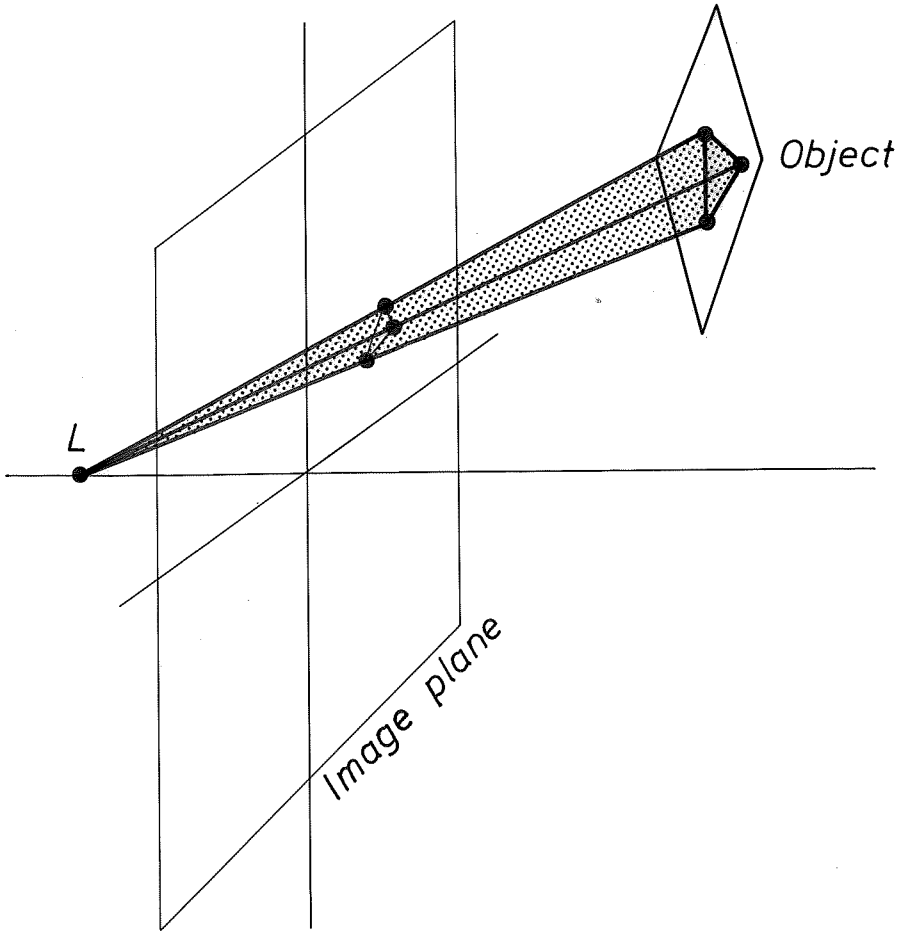


Figure 5.1 An illustration of imaging. A triangle positioned in three dimensions is mapped to the image plane via straight lines through L.

5.1 Imaging

Imaging is illustrated in Figure 5.1. Equations of imaging will be given in this section. Two coordinate systems are introduced. One is fixed to the camera and the other is fixed relative to the object. Compare Figure 5.2. Denote the camera coordinate system with subscript c. Let f be the focal length of the camera. The z_c-axis is oriented along the optical axis, and the position of the camera lens is $(0,0,-f)^T$. This point is denoted L. The image plane is the plane $z_c = 0$. A non-inverted image is thus obtained by thinking the image plane in front of the lens.

Denote the object coordinate system with subscript o. It is here restricted to objects described by a finite set of points e.g. the vertices of a number of

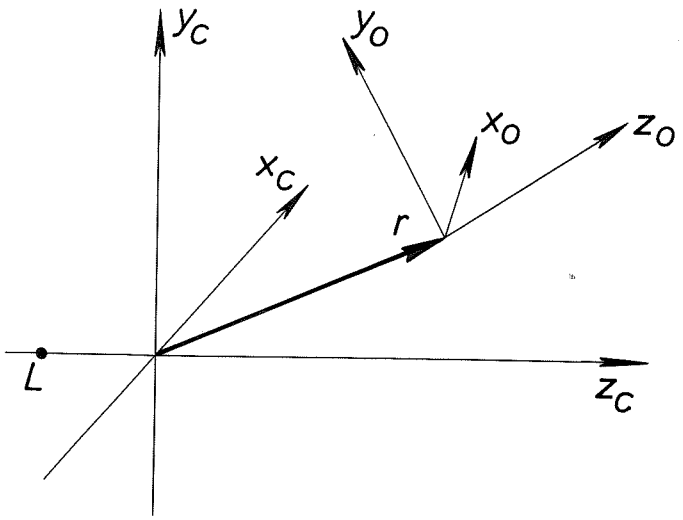


Figure 5.2 An illustration of the relation between the object coordinate system and the camera coordinate system.

polygons. Let the points be given by

$$r_o(j) = \begin{bmatrix} x_o(j) \\ y_o(j) \\ z_o(j) \end{bmatrix} \qquad j = 1,\ldots,n \tag{5.1}$$

The object is characterized by these points combined to the matrix

$$R_o = [r_o(1) \ \ldots \ r_o(n)] \tag{5.2}$$

For planar objects the object coordinates are chosen so that

$$z_o(j) = 0 \qquad \forall j \tag{5.3}$$

i.e. planar objects are parameterized in the xy-plane of the object coordinate system.

Coordinate transformation

The object may be arbitrarily positioned in the camera coordinate system. This is equivalent to an arbitrary orthogonal transformation Q and an arbitrary translation r of the object coordinate system relative to the camera coordinate system. Compare Figure 5.2. Any point r_o of the object is in the camera coordinates described by

$$r_c = r + Q\, r_o$$

Let the translation be

$$r = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{5.4}$$

Introduce a translation matrix of the same dimension as R_o in (5.2).

$$R = [r \ldots r] \tag{5.5}$$

Introduce three Euler angles θ, φ, ψ. The rotation matrix may then be written

$$Q = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{bmatrix}$$

$$\begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.6}$$

Multiplying together in (5.6) results in

$$Q = [q_1, q_2, q_3]$$

where

$$q_1 = \begin{bmatrix} \cos\theta\cos\psi - \sin\theta\cos\varphi\sin\psi \\ \sin\theta\cos\psi + \cos\theta\cos\varphi\sin\psi \\ \sin\varphi\sin\psi \end{bmatrix} \tag{5.7}$$

$$q_2 = \begin{bmatrix} -\cos\theta\sin\psi - \sin\theta\cos\varphi\cos\psi \\ -\sin\theta\sin\psi + \cos\theta\cos\varphi\cos\psi \\ \sin\varphi\cos\psi \end{bmatrix} \tag{5.8}$$

$$q_3 = \begin{bmatrix} \sin\theta\sin\varphi \\ -\cos\theta\sin\varphi \\ \cos\varphi \end{bmatrix} \tag{5.9}$$

The positioning variables are assembled into the vector ξ i.e.

$$\xi = (x,\ y,\ z,\ \theta,\ \varphi,\ \psi)^T \tag{5.10}$$

The coordinates of the points are in the camera coordinate system given by

$$R_c = R + QR_o \tag{5.11}$$

The object coordinate system and the camera coordinate system coincide for $\xi = 0$. Any position of the object can be represented by ξ.

The imaging transformation

The imaging transformation is a mapping from a point A in the camera coordinate system to a corresponding point B in the image plane. Compare Figure 5.3. The image point B is the intersection between the image plane $z_c = 0$ and the line between A and L:$(0,\ 0,\ -\ f)^T$. The point P is $(0,0,z_c(j))^T$. Since, in Figure 5.3, the triangles LOB and LPA are similar it follows that

$$\begin{cases} x_i(j) = f \cdot \dfrac{x_c(j)}{z_c(j) + f} \\ \\ y_i(j) = f \cdot \dfrac{y_c(j)}{z_c(j) + f} \end{cases} \qquad j = 1,\ldots,n \tag{5.12}$$

The image points $r_i(j) = [x_i(j)\ \ y_i(j)]^T$ are composed to the image

$$R_i = [r_i(1)\ \ldots\ r_i(n)] \tag{5.13}$$

It is assumed that the object is in front of the camera, rather than alongside or behind it. This implies that

$$z_c(j) > -\ f \qquad , \forall j \tag{5.14}$$

For a given object, the condition (5.14) gives a restriction on the positioning variables ξ (5.10). Introduce the notation X for the set of possible ξ-values. The set X will be restricted further in Section 5.3.

To conclude: Consider a given object R_o. Equations (5.11) and (5.12) then give its image R_i as a function of R and Q i.e. as a function of ξ in (5.10). Every possible position of the object in space can be represented by ξ, and hence every possible

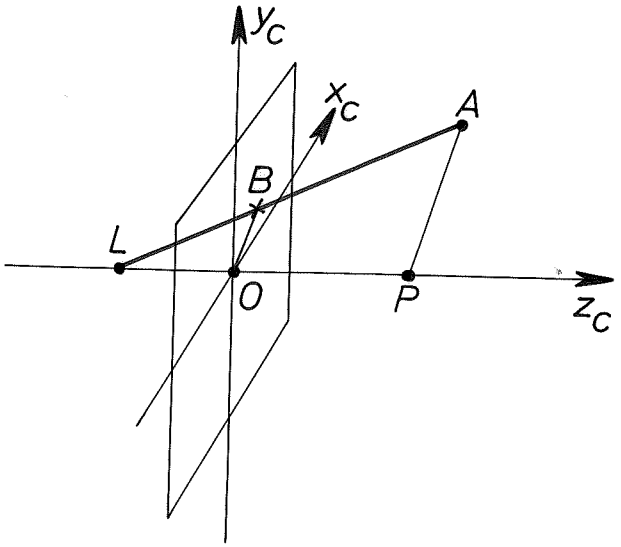


Figure 5.3 The image B of a point A in the camera coordinate system.

image of the object may be represented by ξ. An alternative name for the positioning variables ξ could therefore be the imaging variables.

5.2 Properties of Imaging

Some general properties of imaging will be reviewed in this section. Equations (5.11) and (5.12) have the following properties.

P1. Straight lines map to straight lines (Rosenfeld and Kak, 1982, Chapter 9).

P2. The cross-ratio C between any four points on a straight line is preserved (Rosenfeld and Kak, 1982, Chapter 9). Compare Figure 5.4. The points x_1, x_2, x_3, x_4 on the line ℓ_x are mapped to y_1, y_2, y_3, y_4 on the line ℓ_y via lines through the point L. Then

$$C(y_1, y_2, y_3, y_4) = \frac{(y_4-y_2)(y_3-y_1)}{(y_4-y_1)(y_3-y_2)} =$$

$$= \frac{(x_4-x_2)(x_3-x_1)}{(x_4-x_1)(x_3-x_2)} = C(x_1, x_2, x_3, x_4) \qquad (5.15)$$

The properties P1 and P2 are also fundamental in projective geometry. A

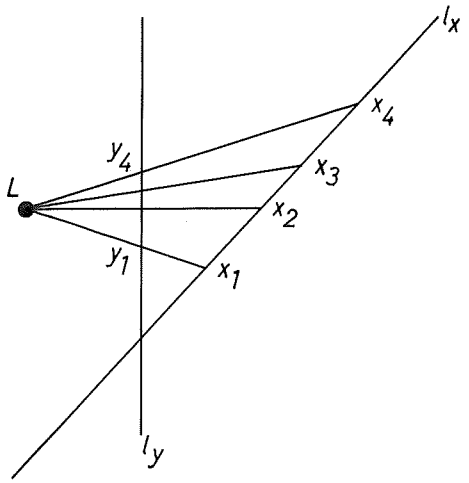


Figure 5.4 An illustration of the imaging of four points on a straight line.

condensed presentation of projective geometry can be found in Coxeter (1963, Chapter 14). The following observations are useful. Firstly, the cross-ratio can be interpreted as a projective coordinate for a fourth point on a line, since a fourth point is uniquely given from three points and the cross-ratio. Secondly, four points may be mapped arbitrarily in a projective mapping from one plane to another one. A consequence is that a square may be mapped to any quadrangle.

It is now important to distinguish between projective mappings and imaging. Imaging may be regarded as a projective mapping subject to the restriction (5.14). This restriction has the consequence

P3. Convexity is preserved by imaging.

Notice, however, that convexity is not preserved by projective mappings. The property P3 is a result of the restriction (5.14) which means that the order between points is preserved and hence convexity. It can also be concluded from Figure 5.4 where only those points of ℓ_x that are to the right of L are mapped to ℓ_y. The property P3 does not seem to have been pointed out before, but it is on the other hand straight-forward to see.

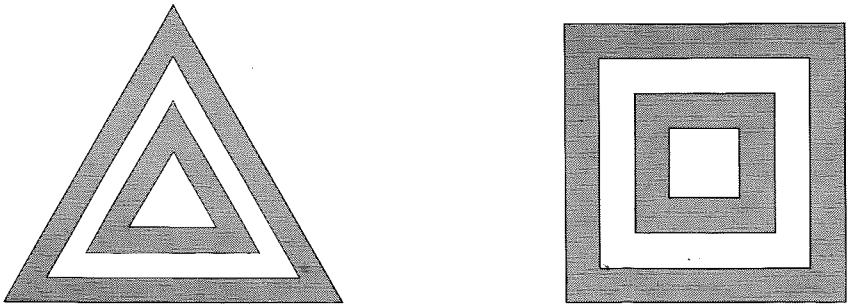


Figure 5.5 The concentric squares, m=4, and triangles, m=3, with k=(1,2,3,4).

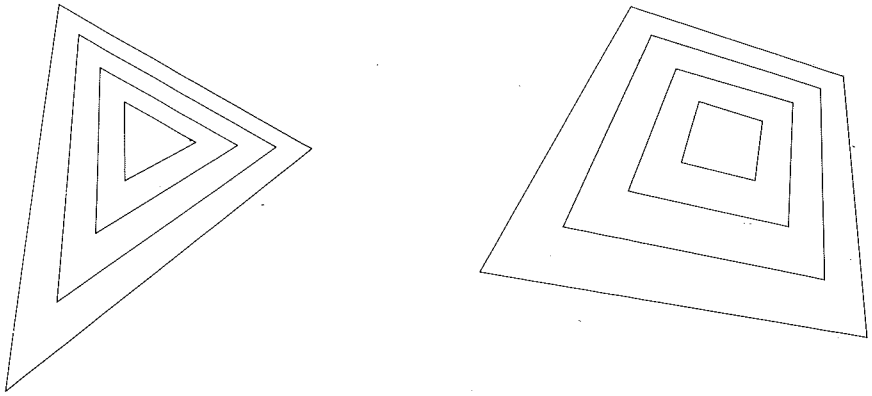


Figure 5.6 Possible images of the contours of the objects in Figure 5.5.

5.3 Problem Formulation

The problem of perspective invariant marking consists of constructing objects and invariants of these objects that can be used to separate different objects.

Objects

We restrict ourselves to study planar symbols composed of triangles or squares. The shapes of images of such objects may be detected using an algorithm in Section 6.2. The two primitive objects are a square and an equilateral triangle. Let m be the number of corners of the primitive object. Composite objects are obtained by n concentric squares and triangles respectively. The objects are for n = 4 illustrated in Figure 5.5. By imaging, (5.11) and (5.12), the objects are mapped to images e.g. as shown in Figure 5.6. The primitive objects are not similar in the image, and parallel lines of the objects are not parallel in the image.

There exist cases where the image of a planar object reduces to a line segment. This happens if the plane of the object goes through the point L:(0,0, - f). In that case the object can not be identified, and hence this singular case will be rejected in the following. Exclude such ξ from X. For planar objects with $z_o = 0$ the vector q_3 (5.9) is the normal of the object expressed in camera coordinates, and we have

$$X = \{ \xi : q_3^T \cdot (r-L) \neq 0 \text{ and } z_c(j) > -f, \forall j \} \qquad (5.16)$$

where the restriction (5.14) is included.

Parameterization

Let the sizes of the primitive objects be parameterized by their circumscribed circles. Let the radii of the n circles be

$$b, k_2 \cdot b, \ldots, k_n \cdot b$$

where the value of b is a scale factor of size. The relative size can thus be represented by the vector

$$k = (k_j) \qquad j = 1,\ldots,n \qquad (5.17)$$

where $k_1 = 1$.

A certain composite object is now uniquely described by

$$\sigma = (m, b, k) \qquad (5.18)$$

To satisfy the restriction (5.14) it is required that

$$z + f > k_n b > k_2 b > k_1 b = b \qquad (5.19)$$

The center of the objects is placed in the origin of the object coordinate system and one corner is denoted $r_o = (b \quad 0 \quad 0)^T$. This leads to a unique parameterization as follows. The corners of the object consisting of n concentric squares, m=4, and the object consisting of n concentric triangles, m=3, may be written as

$$R_o = [r_o(1) \ldots r_o(mn)] \qquad (5.20)$$

where

$$r_o(j+m\ell) = b\, k_{\ell+1} \left[\cos \frac{2\pi(j-1)}{m},\ \sin\frac{2\pi(j-1)}{m},\ 0\right]^T$$

$$j = 1,\ldots,m\ ;\ \ell = 0,\ldots,n-1$$

Requirements

The information obtained from the image processing (Chapter 4) is integrated measures or more general contour attributes. The information is formalized as a vector of feature values

$$a = (a_j) \tag{5.21}$$

The elements a_j could be areas, perimeters, or corner positions. Different cases will be treated. The vector a is of course a function of both the object shape, parameterized by σ in (5.18), and the positioning variables ξ in (5.10).

$$a = a(\xi;\sigma) \tag{5.22}$$

We are now searching a function F of the feature vector, which is invariant for a given object i.e. for a given σ. It is restricted to differentiable functions F. Further F must be such, that it is possible to find different symbols between which F uniquely classifies. The functions F may depend on σ. Recall that X is the set of possible ξ-values excluding those ξ where the image degenerates to a line (5.16). The requirements are

1. Regard a certain symbol σ. F is invariant, if $F(a;\sigma)$ is independent of ξ, i.e. if

$$F(a(\xi_1;\sigma);\sigma) = F(a(\xi_2;\sigma);\sigma) \tag{5.23}$$

   for all ξ_1 and ξ_2 in X.

2. There should exist a set Σ of symbols σ, so that $F(a;\sigma)$ classifies uniquely i.e. let $\sigma, \bar{\sigma} \in \Sigma$. Then it is required that

$$F(a(\xi_1;\sigma);\sigma) = F(a(\xi_2;\bar{\sigma});\sigma) \quad \Rightarrow \quad \sigma = \bar{\sigma} \tag{5.24}$$

   for any $\xi_1, \xi_2 \in X$.

Assume that the type of features in the vector a is given. The problem of finding a set of symbols for marking is solved if an invariant F and a set Σ are found so that (5.23) and (5.24) hold.

5.4 Perspective Invariants Based on Integrated Measures

Consider the concentric squares and triangles (5.20). Let areas within closed curves be the fundamental identification property. Hence the available image information can easily be obtained as MeasureList given in Section 4.2.

The elements of the feature vector are thus areas. We have for n primitive objects n areas in the image of the composite object. These areas are denoted

$$a = (a_j) \qquad j = 1,\ldots,n \tag{5.25}$$

Let a_j be sorted in increasing order. Note that from (5.16)

$$a_j \neq 0 \qquad \forall j \qquad \forall\, \xi \in X \tag{5.26}$$

The problem is now to characterize the functions $F(a;\sigma)$ that fulfill (5.23) and (5.24). First the relation between two areas in the image will be determined. Then equation (5.23) will be solved to obtain all solutions F. Thereafter the resolution power of F to separate between objects is treated.

The area relation

Consider the image of the two smallest squares in (5.20), m=4, i.e. $r_i(j)$, j = 1,...,8. The image may look like in Figure 5.7. The relation between the areas can be obtained by inserting (5.20), for n = 2 and m = 4, into (5.11) and (5.12) and then solving for a_2 as a function of a_1. To obtain insight and to simplify the calculations a discussion based on the properties P1 - P3 of Section 5.2 will be given in parallel.

The four corners on a diagonal are in the image still on a straight line because of property P1. Introduce the lengths of the diagonals

$$\begin{cases} d_1 = |r_i(1) - r_i(3)| \\ d_2 = |r_i(2) - r_i(4)| \\ d_3 = |r_i(5) - r_i(7)| \\ d_4 = |r_i(6) - r_i(8)| \end{cases} \tag{5.27}$$

We have for $\xi \in X$ that $d_j \neq 0$, j=1,2,3,4. The cross-ratio C (5.15) between the four

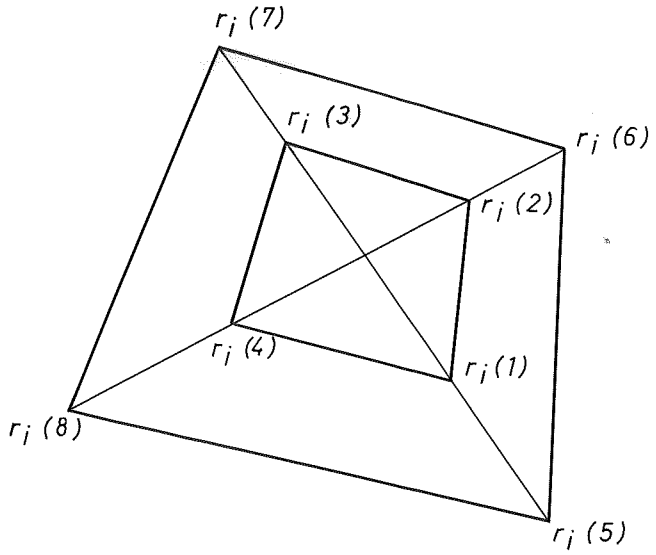


Figure 5.7 A possible image of the object (5.20) for n = 2 and m = 4.

corners on each diagonal is preserved because of property P2. Hence the cross-ratio between $r_i(1)$, $r_i(3)$, $r_i(5)$ and $r_i(7)$ would be determined by e.g. d_1, d_3 and $|r_i(1) - r_i(5)|$. By knowing only d_1 and d_3 we are left with one degree of freedom. Hence d_3/d_1 is a function of only one variable. The explicit expression obtained from (5.11), (5.12) and (5.20) is

$$\frac{d_3}{d_1} = \frac{k_2}{k_1} \cdot \frac{(z+f)^2 - k_1^2 b^2 \sin^2\varphi \sin^2\psi}{(z+f)^2 - k_2^2 b^2 \sin^2\varphi \sin^2\psi} \qquad (5.28)$$

which is a function only of the variable

$$\frac{\sin^2\varphi \sin^2\psi}{(z+f)^2}$$

In the same way we find that

$$\frac{d_4}{d_2} = \frac{k_2}{k_1} \cdot \frac{(z+f)^2 - k_1^2 b^2 \sin^2\varphi \cos^2\psi}{(z+f)^2 - k_2^2 b^2 \sin^2\varphi \cos^2\psi} \qquad (5.29)$$

is a function only of

$$\frac{\sin^2\varphi \cos^2\psi}{(z+f)^2}$$

Now consider the relation between the areas in Figure 5.7. Consider a convex quadrangle with diagonals δ_1 and δ_2. Let γ be the angle between the diagonals. The area of the quadrangle is then

$$\frac{1}{2} \cdot \delta_1 \cdot \delta_2 \cdot \sin \gamma$$

Hence

$$\frac{a_2}{a_1} = \frac{d_3}{d_1} \cdot \frac{d_4}{d_2} \cdot \frac{\sin \gamma}{\sin \gamma} \tag{5.30}$$

Both d_3/d_1 (5.28) and d_4/d_2 (5.29) are functions of one variable, but not the same one. Hence the ratio a_2/a_1 (5.30) is a function of two variables. By inserting (5.28) and (5.29) in (5.30) and simplifying it is found that

$$\frac{a_2}{a_1} = \frac{k_2^2}{k_1^2} \frac{(z+f)^4 - k_1^2 b^2 (z+f)^2 \sin^2\varphi + \frac{1}{4} k_1^4 b^4 \sin^4\varphi \sin^2 2\psi}{(z+f)^4 - k_2^2 b^2 (z+f)^2 \sin^2\varphi + \frac{1}{4} k_2^4 b^4 \sin^4\varphi \sin^2 2\psi} \tag{5.31}$$

The case of triangles is treated by insertion of (5.20) for n = 2 and m = 3 into (5.11) and (5.12). The derivation is given in Appendix C, and it is performed using a computer program, MACSYMA, for symbolic manipulation of algebraic expressions. The result is

$$\frac{a_2}{a_1} = \frac{k_2^2}{k_1^2} \frac{(z+f)^3 - \frac{3}{4} k_1^2 b^2 (z+f) \sin^2\varphi - \frac{1}{4} k_1^3 b^3 \sin^3\varphi \sin 3\psi}{(z+f)^3 - \frac{3}{4} k_2^2 b^2 (z+f) \sin^2\varphi - \frac{1}{4} k_2^3 b^3 \sin^3\varphi \sin 3\psi} \tag{5.32}$$

Equations (5.31) and (5.32) are the fundamental area relations under imaging of (5.20) i.e. concentric squares, m=4, and triangles, m=3, respectively.

Solving for F

The solutions F to (5.23) will now be obtained. The steps in the solution are: introduce new variables, express (5.23) in these variables, and then solve the resulting more simple equations.

By using the expressions (5.31) and (5.32) for area quotients we can express all areas in three new variables $\varsigma = \varsigma(\xi;\sigma)$ instead of $\xi = (x, y, z, \theta, \varphi, \psi)$. Introduce first the notations

$$r = a_1(\xi;\sigma) \qquad\qquad m = 3,4$$

$$s = \begin{cases} \frac{3}{4}\left[\frac{b}{z+f} \cdot \sin\varphi\right]^2 & m = 3 \\ \left[\frac{b}{z+f} \cdot \sin\varphi\right]^2 & m = 4 \end{cases}$$

$$t = \begin{cases} -\frac{1}{4}\left[\frac{b}{z+f} \cdot \sin\varphi\right]^3 \sin 3\psi & m = 3 \\ \frac{1}{4}\left[\frac{b}{z+f} \cdot \sin\varphi\right]^4 \sin^2 2\psi & m = 4 \end{cases}$$

The variables

$$\varsigma = (\varsigma_1, \varsigma_2, \varsigma_3)^T \tag{5.33}$$

are now defined as

$$\varsigma_1 = \frac{1}{r\ (1-s+t)} \tag{5.34}$$

$$\varsigma_2 = -\ s\ \varsigma_1 \tag{5.35}$$

$$\varsigma_3 = t\ \varsigma_1 \tag{5.36}$$

Introduce Z for the set of ς-values generated by the set X of object positions ξ.

It is favourable to introduce the invariant function in terms of inverse areas $u = u(\varsigma;\sigma)$ instead of areas:

$$u_j(\varsigma(\xi;\sigma);\sigma) = \frac{1}{a_j(\xi;\sigma)} \qquad \forall\ \xi \in X \quad j = 1,\ldots,n \tag{5.37}$$

$$G(u;\sigma) = F(a;\sigma)$$

Recall $k_1 = 1$. Using (5.34), (5.35), (5.36), and (5.37) the area relations (5.31) and (5.32) can be written in a unified way as

$$u_j = k_j^{-2}\ \varsigma_1 + \varsigma_2 + k_j^{m-2}\ \varsigma_3 \qquad m = 3,4 \quad j = 1,\ldots,n \tag{5.38}$$

Introduce the nx3 matrix K where the j:th row is

$$[k_j^{-2} \quad 1 \quad k_j^{m-2}] \tag{5.39}$$

We have now arrived at the fundamental formulation

$$u = K\zeta \tag{5.40}$$

where we have the bijection (5.37) between u and a. The matrix K has rank 3. Hence there exists a linear coordinate transformation

$$\begin{aligned} v &= Tu \\ F(a;\sigma) &= G(u;\sigma) = H(v;\sigma) \end{aligned} \tag{5.41}$$

where we chose T so that

$$\begin{aligned} TK &= I \qquad && n = 3 \\ TK &= \begin{bmatrix} I \\ 0 \end{bmatrix} && n > 3 \end{aligned} \tag{5.42}$$

For n > 3 there is some arbitrariness in the choice of T. One explicit choice is given later.

The equation (5.23) for invariants F can equivalently be expressed as

$$\frac{dH(v(u;\sigma);\sigma)}{d\zeta} = 0 \qquad \forall\, \zeta \in Z \tag{5.43}$$

From the chain-rule of differentiation it follows that (5.43) is equivalent to

$$\frac{dH}{dv}\frac{dv}{du}\frac{du}{d\zeta} = \frac{dH}{dv}TK = 0 \tag{5.44}$$

which is

$$\frac{\partial H}{\partial v_1} = \frac{\partial H}{\partial v_2} = \frac{\partial H}{\partial v_3} = 0 \tag{5.45}$$

and these are the only restrictions on H and thus on F.

For n = 3 the only solution to (5.45) and hence to (5.23) is the trivial one

$$\frac{\partial F}{\partial a_1} = \frac{\partial F}{\partial a_2} = \frac{\partial F}{\partial a_3} = 0 \tag{5.46}$$

Requirement 2 (5.24) is then not satisfied.

For $n \geq 4$ the solution to (5.45), and hence to (5.23) is

$$F(a;\sigma) = f(\tilde{v}) \tag{5.47}$$

$$\tilde{v} = (v_4, \ldots, v_n)$$

where f is an arbitrary differentiable function of n - 3 variables. The variable $\tilde{v}$ is an invariant function of a. All other solutions $f(\tilde{v})$ to (5.23) are said to be generated by $\tilde{v}$. The variables $\tilde{v}$ are called basic invariants or generators (Fogarty, 1969). For n concentric squares or n concentric triangles where $n \geq 4$ we can deduce from (5.47) that we have n - 3 basic invariants v_j. The transformation T is somewhat arbitrary. Here we have chosen v_j to be a function of a_1, a_2, a_3, and a_j as follows.

Partition K as

$$K = \begin{bmatrix} K_1 \\ K_2 \end{bmatrix} \qquad \begin{matrix} K_1 & 3x3 \\ K_2 & (n-3)x3 \end{matrix} \tag{5.48}$$

and with T compatibly partitioned

$$T = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} \tag{5.49}$$

with

$$T_{11} = K_1^{-1} \qquad T_{12} = 0$$

$$T_{21} = - T_{22} K_2 K_1^{-1} \qquad T_{22} = \mathrm{diag}(t_j)$$

Introduce the function

$$h(x_1,x_2,x_3) = x_1^2x_2^m - x_1^mx_2^2 + x_2^2x_3^m - x_2^mx_3^2 + x_3^2x_1^m - x_3^mx_1^2 \tag{5.50}$$

We have

$$h(k_1,k_2,k_3)\ K_1^{-1} =$$

$$\begin{bmatrix} k_1^2\,(k_2^2k_3^m - k_2^mk_3^2) & k_2^2\,(k_3^2k_1^m - k_3^mk_1^2) & k_3^2\,(k_1^2k_2^m - k_1^mk_2^2) \\ k_1^2\,(k_2^m - k_3^m) & k_2^2\,(k_3^m - k_1^m) & k_3^2\,(k_1^m - k_2^m) \\ k_1^2\,(k_3^2 - k_2^2) & k_2^2\,(k_1^2 - k_3^2) & k_3^2\,(k_2^2 - k_1^2) \end{bmatrix} \qquad (5.51)$$

The basic invariants $\tilde{v}$ are

$$\tilde{v} = [T_{21}\ T_{22}]\ u = T_{22}\ [-K_2\ K_1^{-1}\quad I]\ u \qquad (5.52)$$

Chosing t_j in T_{22} as

$$t_j = k_j^2\ \ h(k_1,k_2,k_3) \qquad (5.53)$$

we get the following expressions

$$v_j = \frac{L_{1j}}{a_1} + \frac{L_{2j}}{a_2} + \frac{L_{3j}}{a_3} + \frac{L_{jj}}{a_j} \qquad j = 4,\ldots,n \qquad (5.54)$$

where the coefficients L_{ij} are symmetric expressions in k:

$$\begin{aligned} L_{1j} &= -\ k_1^2\ \ h(k_2,k_3,k_j) \\ L_{2j} &= \quad k_2^2\ \ h(k_3,k_j,k_1) \\ L_{3j} &= -\ k_3^2\ \ h(k_j,k_1,k_2) \\ L_{jj} &= \quad k_j^2\ \ h(k_1,k_2,k_3) \end{aligned} \qquad (5.55)$$

Uniqueness

The existence of a set Σ so that the requirement (5.24) is fulfilled will now be investigated. We exploit the basic invariant (5.54) for n=4

$$F(a;\sigma) = \tilde{v} = v_4 \qquad (5.56)$$

The expression (5.56) is invariant for a given symbol σ = (m,b,k). Let m be given as either 3 or 4.

The scale factor of size, b, does not enter (5.56). Hence objects with the same shape representation vector k but having different sizes cannot be distinguished.

Now let the vector k vary. Consider in particular vectors k and $\bar{k}$ that differ in one component only, e.g.

$$\begin{cases} k_\ell \neq \bar{k}_\ell & \ell = 2,\ 3,\ \text{or } 4 \\ \\ k_j = \bar{k}_j & \text{for } (j = 1,\ldots,n) \text{ and } (j \neq \ell) \end{cases} \tag{5.57}$$

Hence Σ is defined by n=4, m=3 or 4, b arbitrary, and k as (5.57), only restricted by (5.19). The left hand side of equation (5.24) is zero for the invariant (5.56) as seen from (5.40), (5.41), and (5.42). Then (5.24) reduces to the requirement that the only solution to

$$v_4(u(\varsigma;\bar{\sigma});\sigma) = 0 \quad \text{is} \quad k_\ell = \bar{k}_\ell \tag{5.58}$$

We have from (5.52) and (5.40)

$$\begin{aligned} v_4(a(\varsigma;\bar{\sigma});\sigma) &= t_4 \left[-K_2 K_1^{-1} \quad I\right] u(\varsigma;\bar{\sigma}) = \\ &= t_4 \left[-K_2 K_1^{-1} \quad I\right] \bar{K}\, \varsigma \\ &= t_4 \left[-K_2 K_1^{-1} \bar{K}_1 + \bar{K}_2\right] \varsigma \end{aligned} \tag{5.59}$$

In the last step $\bar{K}$ is partitioned as in (5.48). Insertion of (5.59) into (5.58) and elimination of non-zero constants gives

$$\left[k_\ell^2 \bar{k}_\ell^m - k_\ell^m \bar{k}_\ell^2\right] \varsigma_3 - \left[\bar{k}_\ell^2 - k_\ell^2\right] \varsigma_1 = 0 \qquad \ell = 2,3,\text{or } 4 \tag{5.60}$$

The equation (5.60) will be solved for m = 3 and m = 4. The variable ς_1 is non-zero for $\forall\ \xi \in X$.

Case m = 3

Equation (5.60) reduces to

$$\left[k_\ell - \bar{k}_\ell\right]\left[k_\ell^2 \bar{k}_\ell^2 \frac{\varsigma_3}{\varsigma_1} - \left[\bar{k}_\ell + k_\ell\right]\right] = 0 \tag{5.61}$$

and (5.19) and (5.36) give the inequalities

$$-\frac{1}{4}\cdot\frac{1}{k_4^3} < \frac{\varsigma_3(\xi;\sigma)}{\varsigma_1(\xi;\sigma)} < \frac{1}{4}\cdot\frac{1}{k_4^3} \tag{5.62}$$

$$-\frac{1}{4}\cdot\frac{1}{\bar{k}_4^3} < \frac{\varsigma_3(\xi;\bar{\sigma})}{\varsigma_1(\xi;\bar{\sigma})} < \frac{1}{4}\cdot\frac{1}{\bar{k}_4^3} \tag{5.63}$$

The only solution to (5.61) and hence to (5.24) is

$$k_\ell = \bar{k}_\ell \qquad \text{i.e.} \quad \sigma = \bar{\sigma} \tag{5.64}$$

if, in the case of $\ell = 4$, we add the requirement

$$\frac{(\sqrt{2}-1)}{2}\,k_4 < \bar{k}_4 < \frac{2}{(\sqrt{2}-1)}\,k_4 \tag{5.65}$$

Case m = 4

The calculation is analogous to the preceding case. Equation (5.60) reduces to

$$\left[k_\ell-\bar{k}_\ell\right]\left[k_\ell^2\bar{k}_\ell^2\,\frac{\varsigma_3}{\varsigma_1} - 1\right] = 0 \tag{5.66}$$

and (5.19) and (5.36) give the inequalities

$$0 < \frac{\varsigma_3(\xi;\sigma)}{\varsigma_1(\xi;\sigma)} < \frac{1}{4}\cdot\frac{1}{k_4^4} \tag{5.67}$$

$$0 < \frac{\varsigma_3(\xi;\bar{\sigma})}{\varsigma_1(\xi;\bar{\sigma})} < \frac{1}{4}\cdot\frac{1}{\bar{k}_4^4} \tag{5.68}$$

The only solution to (5.66) and hence to (5.24) is

$$k_\ell = \bar{k}_\ell \qquad \text{i.e.} \quad \sigma = \bar{\sigma} \tag{5.69}$$

if, in the case of $\ell = 4$, we add the requirement

$$\frac{1}{2}\,k_4 < \bar{k}_4 < 2\,k_4 \tag{5.70}$$

The result can now be summarized. One component of the shape vector is varied. If $\ell = 2$ or $\ell = 3$ then the inequalities (5.19) are sufficient to ensure

unique classification for both m = 3 and m = 4. For $\ell = 4$ the inequalities (5.65) and (5.70) ensure uniqueness.

Summary

Invariants based on areas have been developed for the concentric squares and triangles (5.20). The invariants were obtained by solving equation (5.23). They fulfill the requirement (5.24). One element of the shape vector k (5.17) can be varied in an interval according to (5.19), (5.65) and (5.70). Hence we have shown the existence of a continuum of shapes fulfilling the requirements. Further an area a_j for $j > 3$ is uniquely determined from a_1, a_2 and a_3 and the basic invariant (5.54). The basic invariant may hence be viewed as a perspective area-coordinate in the same sense as the cross-ratio may be viewed as a projective coordinate. Compare Section 5.2.

5.5 Perspective Invariants Based on Contour Attributes

Let the available image information be the positions of corners. The information can thus be obtained from an AttributeList as defined in Section 4.3. The algorithm in Section 6.2 for instance gives the corners of triangles and quadrangles.

Consider an object composed on n squares (5.20), m=4. Denote its image corners by

$$a_j = r_i(j) \qquad j = 1,\ldots,4n \tag{5.71}$$

The vector $a(\xi;\sigma)$ of feature values is thus

$$a = (a_j) \qquad j = 1,\ldots,4n \tag{5.72}$$

A possible image for n = 2 is seen in Figure 5.7. We are now searching an invariant F which satisfies the requirements (5.23) and (5.24). For n = 1 there are no such invariants since four points may be mapped arbitrarily under imaging. For n = 2 there exist possible invariants. Two tractable invariants are the cross-ratios (5.15) along each diagonal, compare Figure 5.7.

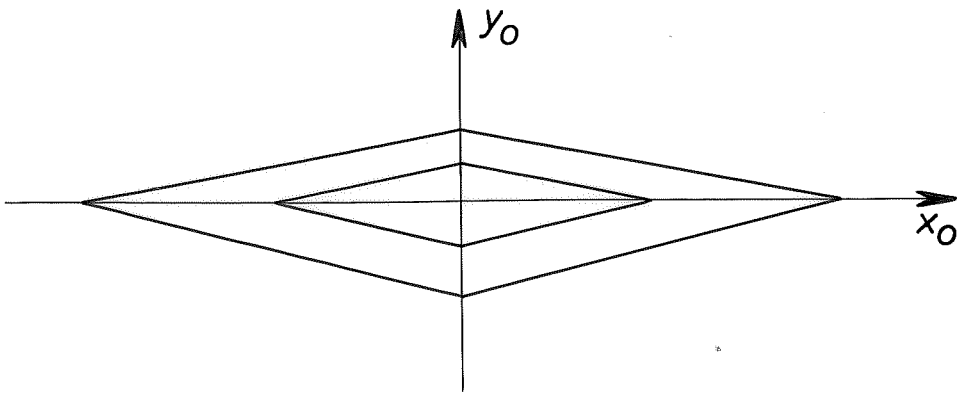


Figure 5.8 An image of two concentric squares.

$$F_1(a;\sigma) = C(a_1,a_3,a_5,a_7) = \frac{|a_7-a_3| \cdot |a_5-a_1|}{|a_7-a_1| \cdot |a_5-a_3|} \tag{5.73}$$

$$F_2(a;\sigma) = C(a_2,a_4,a_6,a_8) = \frac{|a_8-a_4| \cdot |a_6-a_2|}{|a_8-a_2| \cdot |a_6-a_4|} \tag{5.74}$$

For $\xi \in X$ all $|a_i-a_j| \neq 0$, $i \neq j$. The value of F_j is obtained directly from (5.16) as

$$F_1(a;\sigma) = F_2(a;\sigma) = \frac{(k_2b+k_1b)(k_1b+k_2b)}{2\,k_2b \cdot 2\,k_1b} = \frac{(k_1+k_2)^2}{4\,k_1k_2} \tag{5.75}$$

It follows from (5.19) that $k_2 > k_1 = 1$. The right hand side of (5.75) is then strictly increasing with k_2 and hence both F_1 and F_2 classify uniquely between objects of different k_2 i.e. objects of different shapes.

There are advantages of using two invariants if the symbol is viewed in a tilted position like in Figure 5.8. Calculating the cross-ratio under image errors is intuitively more sensitive along the y_o-axis than along the x_o-axis. The longest diagonal should be used for detection.

5.6 Invariants Under Parallel Projection

If the beams from different points of the object to the corresponding image points can be considered parallel, then imaging may be approximated by parallel projection. This case will now be considered. The results are developed in the same way as for perspective invariants in Sections 5.1 - 5.5.

Analytic description

Parallel projection is the special case of imaging where the focal length f is infinite. Hence the coordinate transformation (5.11) is the same but (5.12) and (5.13) is changed to

$$R_i = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} R_c \qquad (5.76)$$

Parallel projection is thus described by (5.11) and (5.76).

Properties

In addition to the properties P1 - P3 in Section 5.2 parallel projection also has the properties

P4. Parallel lines map to parallel lines.

P5. The ratio of division D between any three points on a straight line is preserved. Compare Figure 5.9. The points x_1, x_2, x_3 on the line ℓ_x are mapped to y_1, y_2, y_3 on the line ℓ_y via parallel lines. Then

$$D(y_1, y_2, y_3) = \frac{y_3-y_1}{y_2-y_1} = \frac{x_3-x_1}{x_2-x_1} = D(x_1, x_2, x_3) \qquad (5.77)$$

The parallel projection is, due to (5.14), a restriction of an affine mapping.

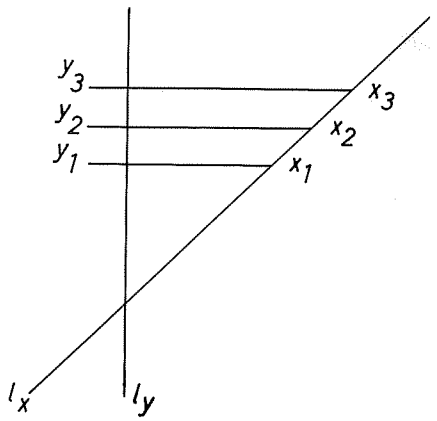


Figure 5.9 Illustrates the parallel projection of three points on a straight line.

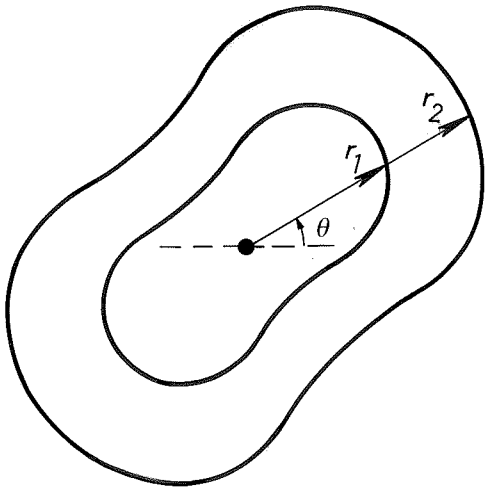


Figure 5.10 Two similar curves with $r_2(\theta) = D \cdot r_1(\theta)$ where D is a constant.

Objects

We will consider objects consisting of two concentric similar curves. Let the inner curve be parameterized as $r_1(\theta)$, see Figure 5.10. The outer curve is given by

$$r_2(\theta) = D \cdot r_1(\theta) \tag{5.78}$$

where D is a constant. Hence the ratio of division is constant. The concentric squares and triangles in (5.20) are special cases of (5.78) with

$$D = k_2 \tag{5.79}$$

Integrated measure invariants

The area inside any closed contours $r_j(\theta)$ in Figure 5.10 is

$$a_j = \frac{1}{2}\int_0^{2\pi} r_j(\theta)^2 \, d\theta \qquad j=1,2 \tag{5.80}$$

and the perimeter is

$$p_j = \int_0^{2\pi} \sqrt{r_j(\theta)^2 + \left[\frac{dr_j(\theta)}{d\theta}\right]^2} \, d\theta \qquad j=1,2 \tag{5.81}$$

The fundamental area- and perimeter-relations are obtained from the fact that the ratio of division (5.77) is invariant. It follows from (5.77), (5.78), (5.79) and (5.80) that

$$a_2 = \frac{1}{2}\int r_2(\theta)^2 \, d\theta = \frac{1}{2}\int k_2^2 \, r_1(\theta)^2 \, d\theta = k_2^2 \cdot a_1$$

or

$$\frac{a_2}{a_1} = k_2^2 \tag{5.82}$$

In the same way it follows from (5.77), (5.78), (5.79) and (5.81) that

$$\frac{p_2}{p_1} = k_2 \tag{5.83}$$

All solutions to the equation (5.23) are obtained using the same technique as in Section 5.4. The fundamental relations here are (5.82) and (5.83). Compare (5.31) and (5.32). Equation (5.23) is solved to obtain all solutions. Compare (5.33) - (5.55). For areas the result is

$$F(a) = f\left[\frac{a_2}{a_1}, \ldots, \frac{a_n}{a_1}\right] \tag{5.84}$$

and for perimeters

$$F(p) = f\left[\frac{p_2}{p_1}, \ldots, \frac{p_n}{p_1}\right] \tag{5.85}$$

where f in both cases is an arbitrary, differentiable function of n - 1 variables.

We deduce from (5.84) and (5.85) that we have n - 1 basic invariants in either case. All possible invariants are generated by these. Further the basic invariants (5.82) or (5.83) classify the objects uniquely. Equation (5.82) may be viewed as an area-coordinate under parallel projection and equation (5.83) may be viewed as a perimeter-coordinate. The result holds for any symbol of the type (5.78) and in particular for concentric squares and triangles.

Corner invariants

The problem formulation of Section 5.5 is reexamined for parallel projection, (5.11) and (5.76), restricted to concentric squares. Consider the diagonals (5.27) in Figure 5.7. For parallel projection it follows from (5.27), (5.78) and (5.79) that

$$\frac{d_3}{d_1} = \frac{d_4}{d_2} = k_2 \tag{5.86}$$

The invariants (5.86) classifies uniquely for $k_2 > 1$. They have the same advantage as the cross-ratio in (5.73), (5.74), and (5.75), when the detection is done in images, where the symbol is viewed in a tilted position like in Figure 5.8. Notice that (5.73), (5.74), and (5.75) may still be used together with (5.86).

5.7 Conclusions

Design of marking symbols has been addressed. The symbols should be possible to identify when they are arbitrarily positioned in three-dimensional space. They can be used for robot marking or as signposts. The key issue is to use invariants under perspective projection, where the transformation is given by (5.11) and (5.12). The identification and the invariants depend on the image information obtained from the image processing.

The concentric triangles and squares (5.20) are sufficient to fulfill the requirements (5.23) and (5.24) in perspective or parallel projection. They can be characterized either by integrated measures or corners. The needed number n of primitive symbols required in (5.20) differ in the different cases. A continuum of symbols can be generated by continuous variation of the relative shape factor k.

The problem formulation is new and so are most results in this chapter. The result of Section 5.4 which leads to area-coordinates under imaging is new. This is an extension of the well-known invariance of the area-quotient (5.82) under parallel projection. Equation (5.82) holds independent of the shape of the areas. We have here shown that there exist invariants based on areas also in the case of perspective imaging. These invariants (5.54) depend on the shape m in (5.20). It also seems to be a new though trivial observation that the perimeter quotient (5.83) for the object (5.78) is invariant under parallel projection.

Finally it should be emphasized that the research on invariants has a direct application in object description. We have used areas, perimeters and corners which are feasible to obtain using current hardware. New approaches to image processing or new hardware will motivate the search for invariants based on new elements. Conversely, the discovery of new invariants may motivate new hardware.

6. Image Processing in the Visual Servo

The image processing in the visual servo is based on the ideas discussed in Chapter 4. It is neither pure nor tuned. The main goal has been to have a flexible image data representation suitable for experiments. The hierarchical representation of the data as well as other information about the implementation was given in Section 3.6.

The image interpretation, which leads to the identification and localization of the symbols in a given set, is treated in Sections 6.1 and 6.2. A number of decision thresholds are also introduced in these sections. Section 6.3 discusses the problem of assigning values to the different thresholds based on an error analysis. The result is a decision strategy. The error analysis in Section 6.3 is also used together with the error analysis of robot motion in Appendix B to design the selection of subimages and to introduce iterative search in the image processing. This is done in Sections 6.4 and 6.5 respectively. The conclusions are given in Section 6.6.

6.1 Image Interpretation

The goal of the image interpretation is to do safe identification and position determination. A set of six marking symbols is used. The Turtle is marked with one symbol. The Ilon car is marked with two symbols. This gives the possibility to determine both position and orientation of the Ilon car directly from the image. One symbol is used as a reference in the start up procedure. This symbol and the two remaining symbols in the set may be used to define stations as discussed in Section 3.5.

The set of marking symbols (5.20), n=2, consists of three versions of two concentric squares (k_2 = 1.57, 2.20, 3.23), and three versions of two concentric triangles (k_2 = 1.58, 2.00, 3.21). The outer dimension of the symbols are approximately the same. Four of the symbols used can be seen in images 18-24 of Figure 3.4. The number of corners m (3 or 4) is used for detection. The area- and perimeter-invariants under parallel projection given by (5.82) and (5.83) are also used. The errors due to the approximation of parallel projection are discussed in Section 6.3.

The general description of the image processing was given in Chapter 4. Some details of the procedures are given below. The whole image or a part of it is interpreted. A contour extraction is performed on the grey-level image. Denote the image by f(x,y). The Roberts operator is used to compute gradients (Rosenfeld and Kak, 1982, Section 10.2). This means calculation of

$$\begin{aligned}(\Delta_+ f)(x,y) &= f(x+1,\ y+1) - f(x,y)\\ (\Delta_- f)(x,y) &= f(x,y+1) - f(x+1,y)\end{aligned} \tag{6.1}$$

i.e. that the image is convolved with

$$\Delta_+ = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \text{ and } \Delta_- = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{6.2}$$

A magnitude of the gradient is computed as

$$g(x,y) = \max\left(|\Delta_+ f|,\ |\Delta_- f|\right) \tag{6.3}$$

The bilevel image representing the contours is obtained directly as

$$B(x,y) = \begin{cases} \text{contour point if } g(x,y) \geq t_b \\ \text{not contour point if } g(x,y) < t_b \end{cases} \tag{6.4}$$

This is a simple contour extraction that normally suffices for extraction of the contours of the marking symbols. The advantage is that the computation is fast on a sequential computer. Typical outputs B(x,y) are presented in images 3, 5, 8, 11, and 19 of Figure 3.4.

The connected components are now extracted from the bilevel image B(x,y), defined in (6.4). The border following algorithm and the border finding algorithm in Rosenfeld and Kak (1982, Section 11.2) are used. The border following algorithm follows one border around, and the version with 8-connectedness is used here. The border finding algorithm scans B(x,y) until a border is found. The border is then followed and stored. The scan is resumed and this is repeated until all borders are found. The result is a list of all connected contours collected in the ContourList introduced in Section 4.3. Each contour is represented as a list of the coordinates of its contour points. No grouping of the contours is done, so the interpretation relies on that the symbols give closed connected contours.

The ContourList is processed to obtain the AttributeList. A crude screening is made first. Small contours are disregarded during this step. Each contour is then processed by the shape classifier described later in Section 6.2, and the result of that is that they are classified as a triangle, a quadrangle or something else. The AttributeList contains only those contours that are successfully classified as triangles or quadrangles. Position of the contour centroid, area, perimeter, type of shape and position of corners are stored in the list.

The object identification is performed using the AttributeList. The symbols are composed of either two triangles or two quadrangles with the same center. It is decided that a symbol is found if a pair of elements in the AttributeList passes the following tests. Let the subscripts 1 and 2 denote the two elements of AttributeList matching a known symbol denoted by the subscript 0.

$$m_1 = m_2 = m_0 \tag{6.5}$$

$$|\text{centroid}_1 - \text{centroid}_2| < t_c \tag{6.6}$$

$$\left|\frac{a_2}{a_1} - k_{2,0}^2\right| < k_{2,0}^2 \cdot t_a \tag{6.7}$$

$$\left|\frac{p_2}{p_1} - k_{2,0}\right| < k_{2,0} \cdot t_p \tag{6.8}$$

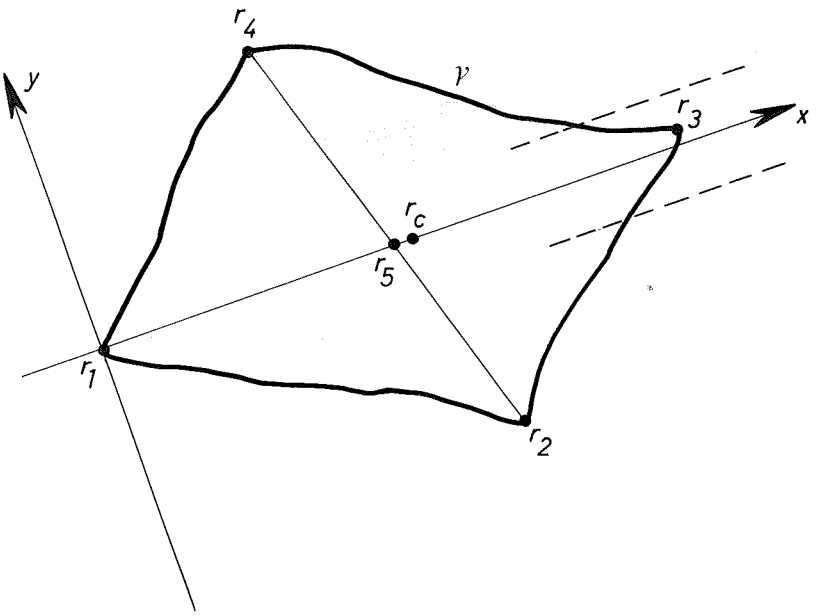


Figure 6.1 An illustration of the case where γ is a closed contour.

6.2 A Shape Classifier

The goal of the shape analysis is to classify a set of points as a triangle, a quadrangle or something else. The set of points normally represents a closed contour, or a number of line segments if the contour extraction has somewhat failed. The algorithm used will be illustrated on a closed contour. It will, however, work on any set of points. The set of points is denoted by γ. A problem that is not treated here is how to correctly group broken contour segments.

A two step algorithm is used. The corners are central for the algorithm. The first step leads to an hypothesis about the number of corners together with rough estimates of their positions. Recall that a triangle has m = 3 and a quadrangle has m = 4. For other objects we set m = 0. Six points r_c, r_1, r_2, r_3, r_4 and r_5 are computed as shown in Figure 6.1. The centroid r_c and the point r_1 in γ which is most distant from r_c are computed first. The point r_c is used as the position of γ. An orthogonal coordinate-system is defined as in Figure 6.1. The x-axis goes through r_c and r_1. The point r_1 is chosen as origin. The extremum-values of γ in this coordinate-system are then computed. The points r_2, r_3 and r_4 corresponds to $y_2 = y_{min}$, $x_3 = x_{max}$ and $y_4 = y_{max}$ respectively. The value x_{max} is computed within certain limits as indicated by the broken lines

in Figure 6.1. Finally the point r_5 is the intersection between the x-axis and the line through r_2 and r_4.

A hypothesis about the shape m_H is now formed as

$$m_H = \begin{cases} 3 & \text{if} \quad |r_3-r_1| < t_q \, |r_5-r_1| \\ 4 & \text{if} \quad |r_3-r_1| \geq t_q \, |r_5-r_1| \end{cases} \tag{6.9}$$

Ideally we have $|r_3-r_1| = |r_5-r_1|$ for a triangle and $|r_3-r_1| = 2\,|r_5-r_1|$ for a parallelogram.

A first estimate of the corner positions is given by

$$\begin{cases} r_1, r_2, r_4 & \text{if} \quad m_H = 3 \\ r_1, r_2, r_3, r_4 & \text{if} \quad m_H = 4 \end{cases} \tag{6.10}$$

The second step in the analysis classifies the shape into one of the three possibilities. A more accurate determination of the corner positions is made in the case of a triangle or a quadrangle. The corner estimates (6.10) are first used to divide γ into m_H subsets γ_j. The intention is that the different γ_j:s should correspond to the different sides of the object. For a closed contour the subsets γ_j are taken as the curves between the approximate corners (6.10) using the ordering of the list. For a general set of points γ the following method may be used. Take m_H pairs of the approximate corners r_j (6.10) consecutive in index. A distance between a point of γ and a pair of approximate corners is the sum of the two distances between the point and the two corners. The subsets γ_j are now formed by the points that have the smallest distance to a certain pair compared to the other pairs.

A straight line

$$\ell_j : a_j x + b_j y + c_j = 0 \tag{6.11}$$

is then fitted to each γ_j using linear regression. Different tests can be used to determine whether γ_j can be considered as a straight line. One possibility could be to use the regression errors in (6.11). Instead we require that all points (x_p, y_p) of γ_j should be within the distance t_ℓ to ℓ_j i.e. that

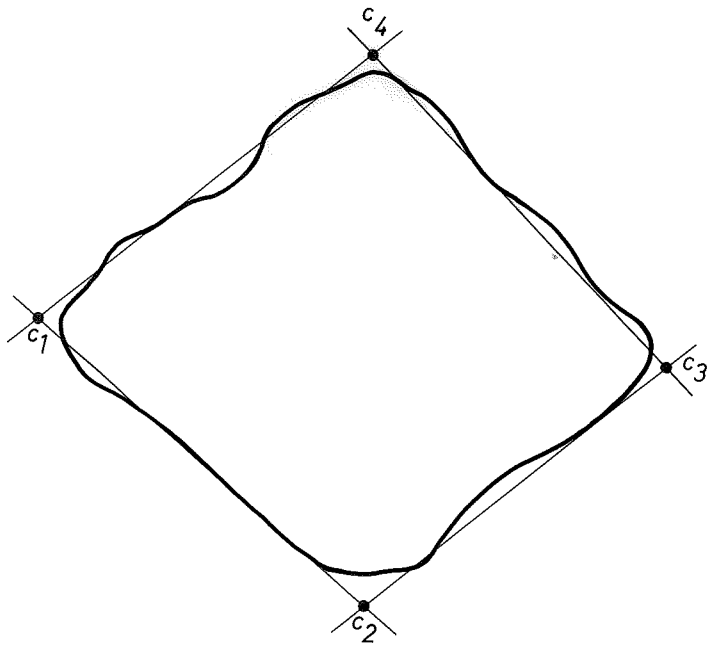


Figure 6.2 It is tested if γ is composed of roughly straight lines. In that case accurate corner positions c_j are determined as the intersections of the lines.

$$\frac{|a_j x_p + b_j y_p + c_j|}{\sqrt{a_j^2 + b_j^2}} < t_\ell \qquad (6.12)$$

for all $(x_p, y_p) \in \gamma_j$. If this is satisfied we write $S(\gamma_j) = \text{line}$.

The shape of γ is now classified as

$$\begin{cases} m = 3 & \text{if } m_H = 3 \text{ and all } S(\gamma_j) = \text{line} \\ m = 4 & \text{if } m_H = 4 \text{ and all } S(\gamma_j) = \text{line} \\ m = 0 & \text{if not all } S(\gamma_j) = \text{line} \end{cases}$$

A more accurate determination of the corner positions is made if γ is classified as either m = 3 or m = 4. The corners are determined as the intersection of the lines ℓ_j as illustrated in Figure 6.2. The Figure 6.2 also illustrates how the algorithm handles rounded corners. The corners may be rounded due to some lack of acuity of the imaging system, see image 11 in Figure 3.4.

6.3 Decision Thresholds

The decision thresholds of (6.4), (6.6), (6.7), (6.8), (6.9), and (6.12) namely

$$t = (t_b, t_c, t_a, t_p, t_q, t_\ell) \tag{6.13}$$

have to be given values. First a crude error analysis is performed to obtain estimates of the maximal errors due both to perspective and to noise. The decision strategy is then given.

The intention of the following error analysis is to get safe thresholds t. The analysis is done for the Turtle symbol. It has been experimentally verified that the values obtained are suitable both for the Turtle symbol and the other symbols used.

Errors of perspective

The tests (6.7), (6.8) and (6.9) with thresholds t_a, t_p and t_q are based on the idea of parallel projection. To what extent is the approximation valid for the symbol of the Turtle?

The experimental dimensions are given in Section 3.1. Compare also Figure 3.1. The restriction that the Turtle moves on the floor will be described in the camera coordinates introduced in Section 5.1. The equation of the floor plane is

$$-0.84\,y + 0.53\,z - 1.85 = 0 \tag{6.14}$$

The restrictions in position and orientation, can be expressed as

$$\begin{aligned} &2.5 < (z+f) < 5.1 \\ &|x| < 0.95 \\ &\theta = 0 \\ &\varphi = 58° \\ &\psi \text{ arbitrary} \end{aligned} \tag{6.15}$$

The restriction in y follows from equation (6.14). The Turtle symbol has two concentric squares (5.20) with

$$\begin{cases} b = 0.05 \\ k_2 = 2.2 \end{cases} \tag{6.16}$$

The Turtle symbol (6.16) undergoes imaging (5.11) and (5.12) under the constraints (6.14) and (6.16). This gives the inequalities

$$1.00\ k_2^2 \leq \frac{a_2}{a_1} < 1.01\ k_2^2 \tag{6.17}$$

$$1.00\ k_2 \leq \frac{p_2}{p_1} < 1.01\ k_2 \tag{6.18}$$

where (6.17) is obtained directly from (5.31) and (6.18) is obtained numerically. The parallel projection approximation is well motivated here.

The test (6.9) will now be examined. Regard the perspective image of a quadrangle, Figure 5.7. Introduce $r_i(0)$ for the intersection between the diagonals and δ_q as the ratio

$$\delta_q = \frac{|r_i(1) - r_i(3)|}{|r_i(1) - r_i(0)|}$$

Imaging (5.11) and (5.12) gives

$$\delta_q = 2 \cdot \frac{z+f}{z+f - b \sin\varphi \sin\psi} \tag{6.19}$$

The restrictions (6.15) and (6.16) give

$$1.96 < \delta_q < 2.04 \tag{6.20}$$

Bias

There are many imperfections in the image data. The two main errors in γ compared to an ideal image of the Turtle symbol are rounding of corners and a bias towards brighter areas in edge position. Rounding of corners means that the connection between two sides is a smooth curve rather than a sharp vertex. The effect can be seen in image 11 of Figure 3.4. This problem can be neglected because the second step of the shape classifier compensates for it as illustrated in Figure 6.2. The bias in edge position can be seen by comparing Figure 3.2 with image 3 of Figure 3.4.

Both the perimeter and the area of the inner quadrangle will be overestimated because of the bias in the edge detection. The outer quadrangle on the other hand

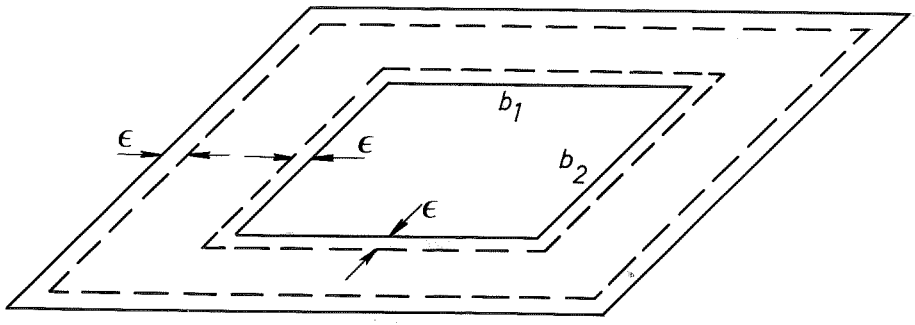


Figure 6.3 The Turtle symbol under parallel projection. The broken lines illustrate the effect of the bias in the edge detection.

is underestimated. A rough estimate of the maximal resulting relative errors are obtained as follows. Assume parallel projection. Compare Figure 6.3. Let the inner square be mapped to a parallelogram with sides b_1 and b_2. Denote the bias with ε. The bias changes the sides of the parallelogram by the distance ε. The area- and perimeter-quotients are then

$$\frac{a_2}{a_1} = \frac{\left[k_2 b_1 - 2\varepsilon\right]\left[k_2 b_2 - 2\varepsilon\right]}{\left[b_1 + 2\varepsilon\right]\left[b_2 + 2\varepsilon\right]} \tag{6.21}$$

$$\frac{p_2}{p_1} = \frac{k_2 b_1 - 2\varepsilon + k_2 b_2 - 2\varepsilon}{b_1 + 2\varepsilon + b_2 + 2\varepsilon} \tag{6.22}$$

The expressions (6.21) and (6.22) degenerate to (5.82) and (5.83) for $\varepsilon = 0$. Introduce the relative errors δ_a and δ_p in (6.21) and (6.22) as

$$\delta_a = \left[\frac{a_2}{a_1} - k_2^2\right] \cdot \frac{1}{k_2^2} \tag{6.23}$$

$$\delta_p = \left[\frac{p_2}{p_1} - k_2\right] \cdot \frac{1}{k_2} \tag{6.24}$$

Let pixel size be the length unit. In the experiment we have roughly $\varepsilon \leq 1$ and $b_j > 20$. Then

$$|\delta_a| < 0.25 \tag{6.25}$$

$$|\delta_p| < 0.13 \tag{6.26}$$

Note that the limit on δ_p is smaller than on δ_a. Form δ_a/δ_p to investigate this.

The minimum value of δ_a/δ_p with respect to b_1 and b_2 is obtained for $b_1 = b_2 = b$. Then

$$\frac{\delta_a}{\delta_p} = 2 \cdot \frac{1+x\cdot(k_2-1)/k_2}{1+2x} \qquad (6.27)$$

where

$$|x| = \left|\frac{\varepsilon}{b}\right| < 0.05 \qquad (6.28)$$

Hence

$$\delta_a > 1.87\ \delta_p \qquad (6.29)$$

The expressions (6.27), (6.28) and (6.29) explain an experimental finding. The errors in the perimeter-quotient were smaller than the errors in the area-quotient. This is the normal situation if the main error is a bias in the position of the contour.

Decision strategy

The actual choice of the threshold values is called the decision strategy. The goal is to identify the marking symbols. The different thresholds will now be treated one by one. The threshold t_q in (6.9) is used to separate between triangles and quadrangles. The value of the test-quantity is ideally 1 for a triangle and 2 for a parallelogram. Based on the error estimate in (6.20) we find that the value of t_q should be between 1 and 1.96. We use $t_q = 1.5$. The thresholds of edge detection t_b determines if a point is classified as a contour point or not. The philosophy in the visual servo is to keep t_b low so that hopefully all contour points are accepted and closed connected contours are obtained for the symbols. A low t_b has also the consequence that more noise points will also be accepted and it may increase the bias. The threshold t_b is varied if needed, see Section 6.5. The nominal value of t_b is 40, which should be related to the image dynamic 0-255.

The positions of the centroids depend on discretizing effects. A fix t_c which corresponds to half the diagonal of the inner symbol is used in (6.6). The contour segments γ_j are normally fairly close to straight lines. The rounding of corners may, however, cause a few points at the ends of γ_j to give a larger value of the test (6.12). The threshold t_ℓ which governs this is chosen to be 3

i.e. no point of a contour segment is allowed to be more than 3 pixels away from the line. The errors in the tests on area- and perimeter-quotient (6.7) and (6.8) are a combination of perspective errors given by (6.17) and (6.18) and bias errors given by (6.25) and (6.26). We use the values $t_a = 0.25$ and $t_p = 0.13$.

To summarize, estimates of maximal errors are used for threshold selection. This means it has been chosen to be tolerant on each single test. The combination of tests is still safe.

6.4 Selection of Subimages

All image data is not always processed. Only a rectangular subimage is interpreted when the Turtle is tracked during path following. The purpose is computational efficiency. The subimage is chosen as follows. The predicted position of the marking symbol of the Turtle is used as the midpoint of the subimage. The prediction is computed from the last measured position and the move command sent to the Turtle. The shape of the subimage is rectangular, see images 8 and 11 in Figure 3.4. The size of the rectangle is determined by a length scale S, which depends on the predicted distance r from the camera as

$$S = S_0 \cdot \frac{r_0}{r} \tag{6.30}$$

The reference length scale S_0 of the subimage is chosen as the sum of the symbol image size and the effect of robot motion errors both evaluated at a reference distance r_0.

6.5 Iteration in Image Processing

The purpose of the image interpretation is to identify the symbols. What can be done if a symbol is not found? The data hierarchy used in the image interpretation (the grey level image, the bilevel image, the list of contours, the list of contour attributes and the list of found symbols) is stored as mentioned in Section 3.6. If an identification has failed it is then possible to backtrack and redo different steps of the algorithm with modified parameters or rules.

Here it is assumed that identification fails either if the contour is not properly extracted or if the symbol is not within the subimage. The latter may occur due to large motion disturbances. A contour can be broken due to image noise. The other parts of the image interpretation are robust under the experimental conditions described here.

The threshold of edge detection t_b (6.4) and the size of the subimage S (6.30) are alternately modified. The iteration loop is

```
Measure(position) is
  repeat
    SubimageSelection
    SetImageProcessingParameters
    ImageInterpretation
  until RobotPositionFound
```

The size S from (6.30) is doubled in the first iteration but t_b is kept as 40. The next two iterations use the same S but t_b is reduced to 35 and 30 in purpose to get more contour points. If the interpretation fails then the whole image is processed with t_b = 40. The values t_b = 35, 30 are then tried. An alarm is sent if the symbol is not found during these iterations.

6.6 Conclusions

This chapter presents the image processing used in the visual servo. The shape classifier described in Section 6.2 is a new contribution. The experiments have shown it to be simple and safe. The error analysis in Section 6.3 illustrates the choice of decision thresholds. Some of the results are surprising at first sight. We found e.g. that the perimeter-quotient (6.8) is a sharper test than the area-quotient (6.7). The idea of subimage selection based on predictions and the idea of backtracking in the image processing have been illustrated. These algorithms are tuned based on a rough error analysis of both motion errors and image errors.

7. Optimal Digitization of 2-D Images

This chapter is composed from an internal report and a publication in IEEE Trans. on ASSP (Nielsen et al, 1983 and 1984). The condensed derivation of the results is from the paper and the description of the experiments are from the report.

Handling large amounts of data is a general problem in a visual servo. An image may typically be 512x512 pixels with 8 bits per pixel. The possibility to represent an image with less data decreases the computational cost. On a conventional computer the computing time decreases. For special purpose parallel hardware the computer size needed is reduced. Results are given on image representation with MxN pixels and b bits per pixel. The constraint of a fixed total number of bits is treated. The answers to problems of this type obviously depend on the type of image and the information in it. Reasonable assumptions are made so that the optimization problem has a closed form solution. The analytical solution depends on image characteristics. The image characteristics are expressed in terms of intensity range and second order moment of partial derivatives. Special hardware for real-time calculation of moments of functions exist (Anderson, 1984). The closed form solution can then be to adapted, in real-time, e.g. to changes in illumination.

The problem of optimal digitization of 2-D images has been sporadically mentioned in several texts, but it has not been addressed in full details (Pavlidis, 1982, p. 39; Rosenfeld and Kak, 1982, p. 111; Huang et al, 1967). Experimental investigations of the effect of coarse scan/fine print for bilevel images was initiated by Abdou and Wong (1982). No theory was, however, given in this work. Steiglitz (1966) has presented a detailed theory of transmission of an analog signal over a fixed bit-rate channel in the 1-D case. This work has motivated the extension to the 2-D case given in this chapter. The definition

seems to be meaningful in practical applications. It differs from Steiglitz work in the respect that our problem formulation admits an analytical solution.

Having introduced the optimal definition, the theoretical formulation of the problem is discussed in Section 7.1. The optimal solution is derived in Section 7.2. The theoretical studies are illustrated in Section 7.3 by a computer experiment which gives insight into the properties of the optimization procedure. These experiments are also used to verify that the optimization algorithm is somewhat suitable for this problem. The results are summarized in the conclusions in Section 7.4.

7.1 Problem Formulation

Let the original image $f(x,y)$ be a function defined on $\Omega = [0, L_x] \times [0, L_y] \subset R^2$ with values in $V = [\min f, \max f] \subset R^+$. The image is sampled to give a sampled image $\hat{f}(x_i,y_j)$ defined on a M x N rectangular grid G with values in $V \subset R^+$. Ideal sampling is assumed, i.e. $\hat{f}(x_i,y_j) = f(x_i,y_j)$ for $x_i,y_j \in G$. In addition to sampling the values of $\hat{f}(x_i,y_j)$ are also quantized so that $V \subset R^+$ is represented by b bits in 2^b quantization levels. The quantization of $\hat{f}$ is denoted by $Q\hat{f}$, which is the digitized image defined on a M x N grid with discrete values. We want to reconstruct the new function $\tilde{f}$ defined on Ω with values in V. From the function $Q\hat{f}$ the function $\tilde{f}$ can be obtained using many different interpolation schemes (Pratt, 1978; Abdou and Wong, 1982; Steiglitz, 1966).

The optimal digitization problem can be formulated as follows: Assume that the image is represented by a fixed number of bits.

$$M \cdot N \cdot b = C \tag{7.1}$$

Determine M, N, b such that the following error is minimum.

$$E = \iint_{\Omega} [f(x,y) - \tilde{f}(x,y)]^2 \, dxdy \; / \; \iint_{\Omega} dxdy \tag{7.2}$$

A number of restrictions are introduced to make the problem tractable analytically. The function f is characterized by its value range R = max f - min f, and the mean fluctuation rates σ_x and σ_y defined by

$$\sigma_x^2 = \overline{(f'_x)^2} \quad \text{and} \quad \sigma_y^2 = \overline{(f'_y)^2} \tag{7.3}$$

The quantization error n is defined as $n = Q\hat{f} - \hat{f}$. Zero order hold interpolation is used. This means that $\tilde{f}(x,y) = Q\hat{f}(x_i,y_j)$ for x,y around x_i,y_j.

7.2 Solution

The criterion (7.2) is expanded by dividing the image in M x N cells with sides $\delta_x = L_x/M$ and $\delta_y = L_y/N$. The cell midpoint x_i,y_j belongs to the grid G. The contributions from all cells are then summed up. Insertion of the digitization and the interpolation schemes in the criterion (7.2) gives the mean square error

$$\bar{E} = \frac{1}{L_x L_y} \sum_{i,j} \{ \iint_{\square} \overline{[f(x-x_i,y-y_j) - f(x_i,y_j)]^2}\, dxdy + \\ + \iint_{\square} \overline{n(x_i,y_j)^2}\, dxdy \} \tag{7.4}$$

where $\square$ denotes integration over one cell. Steiglitz calls the first term in (7.4) the reconstruction error. This error depends only on the grid resolutions δ_x and δ_y. A Taylor series of f gives the following expression for the reconstruction error in one cell.

$$\frac{1}{12} \cdot (\delta_x^3 \delta_y \sigma_x^2 + \delta_x \delta_y^3 \sigma_y^2) \tag{7.5}$$

The second term in (7.4) is the quantization error which depends only on the number of quantization levels 2^b. Assuming equidistant quantization with the grain $\delta = R \cdot 2^{-b}$ the quantization error is approximated by $\delta^2/12$ and

$$\iint_{\square} \overline{n(x_i,y_j)^2}\, dx\, dy = \delta_x \delta_y \overline{n^2} = \delta_x \cdot \delta_y \cdot \frac{1}{12} \cdot \frac{R^2}{2^{2b}} \tag{7.6}$$

The following formula is then obtained for the total mean square error (7.4).

$$\bar{E} = \frac{1}{12} \cdot \left[\frac{L_x^2 \sigma_x^2}{M^2} + \frac{L_y^2 \sigma_y^2}{N^2} + \frac{R^2}{2^{2b}} \right] \tag{7.7}$$

The optimal digitization problem is to minimize (7.7) subject to the constraint

(7.1). The variables L_x, σ_x, L_y, σ_y, and R are known constants which depend on the image. The problem is solved simply by completing the squares in (7.7) and inserting the constraint (7.1).

The solution is

$$b = \frac{1}{2 \ln 2} \ln \left[C \cdot \ln 2 \cdot \frac{R^2}{L_x \sigma_x \, L_y \sigma_y} \right] \tag{7.8}$$

$$M = \sqrt{\frac{L_x \sigma_x}{L_y \sigma_y}} \cdot \sqrt{\frac{C}{b}} \tag{7.9}$$

$$N = \sqrt{\frac{L_y \sigma_y}{L_x \sigma_x}} \cdot \sqrt{\frac{C}{b}} \tag{7.10}$$

It is interesting to see how the relation between the value range R and the fluctuation rates σ_x and σ_y influence the solution. More fluctuations leads to fewer bits (lower b) and more image resolution. Less fluctuations requires more bits and fewer samples. This agrees with earlier subjective tests (Rosenfeld and Kak, 1982; Huang et al, 1967). The formulas for M and N also explicitly gives the tradeoff between sampling rates in respective direction due to fluctuation in each direction.

7.3 The Experiments

The criterion (7.2) was chosen largely for mathematical convenience. A number of experiments have been carried out to see if the optimum digitization obtained correspond to the subjective notation of a good digitization. Figures 7.1 and 7.2 illustrate two cases of an original image together with a number of digitized versions. The product $C = M \cdot N \cdot b$ cannot be kept constant, since M, N, and b are integers. The hardware also limits the possible values on M and N to 512, 256, 170, 128, 102, 85... (=trunc. (512/k)).

Table 7.1 The rotated triangle wave image experiment. The used numbers M, N, b, the total of bits C, and the resulting value of the criterion E.

Image	M	N	b	C	E ($\cdot 10^{-3}$)
1 A	Original test image				
1 B	256	256	1	65536	16.5
1 C	170	170	2	57800	4.69
1 D	128	170	3	65280	1.88
1 E	128	128	4	65536	1.36
1 F	102	128	5	65280	1.48
1 G	85	102	7	60690	2.12

Rotated triangle wave image

An image is computer generated by rotating, around the image midpoint, a triangle wave with period T and amplitude R/2. This image is seen in 7.1 A. The number of bits used for digitization is $C = [256]^2 = 65536$. The image characteristics are calculated from the function by integrating the square of the derivatives

$$\sigma_x = \sigma_y = \sqrt{2} \cdot \frac{R}{T}$$

and hence

$$\frac{R^2}{L_x\sigma_x \cdot L_y\sigma_y} = \frac{1}{2} \cdot \frac{T}{L_x} \cdot \frac{T}{L_y} .$$

The following numbers were used in the experiments

$$T = 0.12\, L_x = 0.12\, L_y$$

Authors' image

An image of the authors Nielsen et al (1983) was used in the second experiment. The original image is seen in 7.2 A. A scan of the original image gives the characteristics

$$R = 4.06 \cdot 10^{-2}\, L_x\sigma_x = 9.68 \cdot 10^{-2}\, L_y\sigma_y \qquad (7.11)$$

It is seen from the expressions (7.11) that there are more fluctuation horizontally than vertically. The number of bits used for digitization is $C = [256]^2 = 65536$.

Table 7.2 The authors image experiment. The used numbers M, N, b, the total of bits C, and the resulting value of the criterion E.

Image	M	N	b	C	E ($\cdot 10^{-3}$)
2 A	Original authors' image				
2 B	256	256	1	65536	16.0
2 C	256	128	2	65536	4.81
2 D	170	102	3	52020	2.88
2 E	170	85	4	57800	2.44
2 F	128	102	4	52224	3.15
2 G	170	73	5	62050	2.58
2 H	128	64	7	57344	3.89

The integers which are closest to the optimal are M = 170, N = 85 and b =N 4 (Image 7.2 E). If fewer bits/pixel (b < 4) are used the image looks blurred (Image 7.2 B-D). If more bits/pixel (b > 4) and fewer sampling points are used some detail is lost (Image 7.2 G-H). If the optimal number of bits (b = 4) are used but less care is taken to the two directions (M = 128 and N = 102) it gives an "edgier" image because some detail is lost horizontally but not much is gained vertically (Image 7.2 F).

7.4 Conclusions

A theoretical formulation of the optimal digitization problem is given. The solution is obtained by optimizing a criterion with a constraint on the total number of bits used to represent the image. The solution has been tested experimentally and it agrees well with human visual perception of picture quality. An advantage is that the solution is given in closed form. This makes it easy to use. The analytical solution also clearly shows the dependence on image characteristics. This explains results of other subjective tests. The image characteristics can be estimated using existing special purpose hardware. The simplicity of the solution makes real-time adaptation possible. The criterion (7.2) is one of the simplest which admits an analytic solution. It would be interesting to look at other alternatives and make more extensive experimentation.

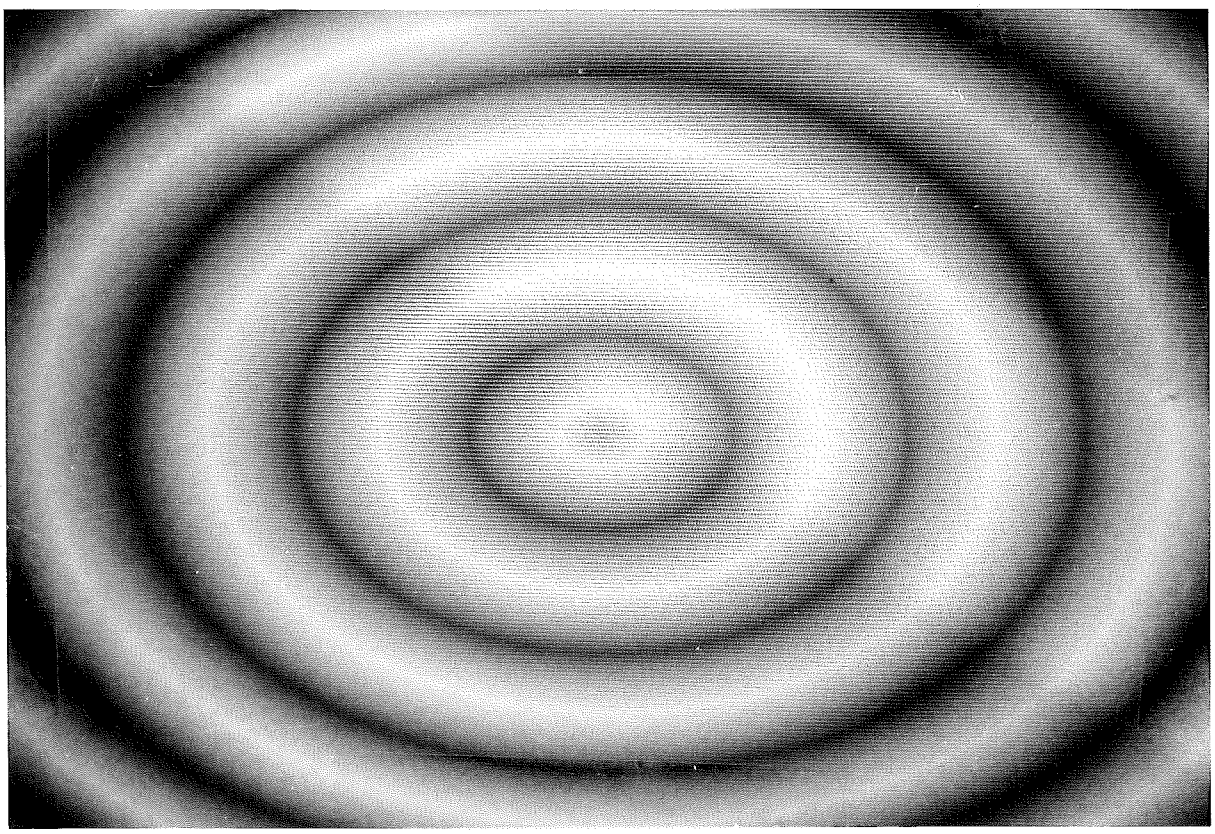


Image 7.1 A. Original test image

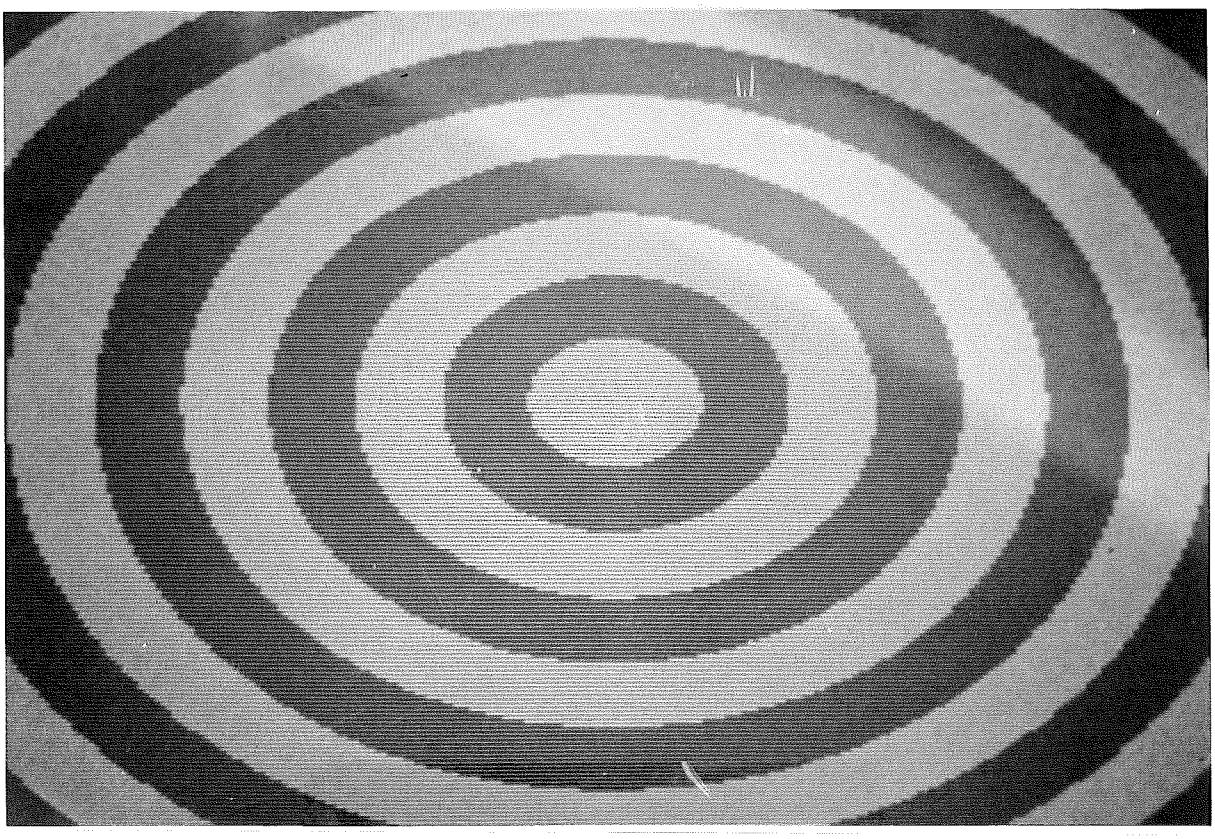


Image 7.1 B. M=256 N=256 b=1 M·N·b=65536

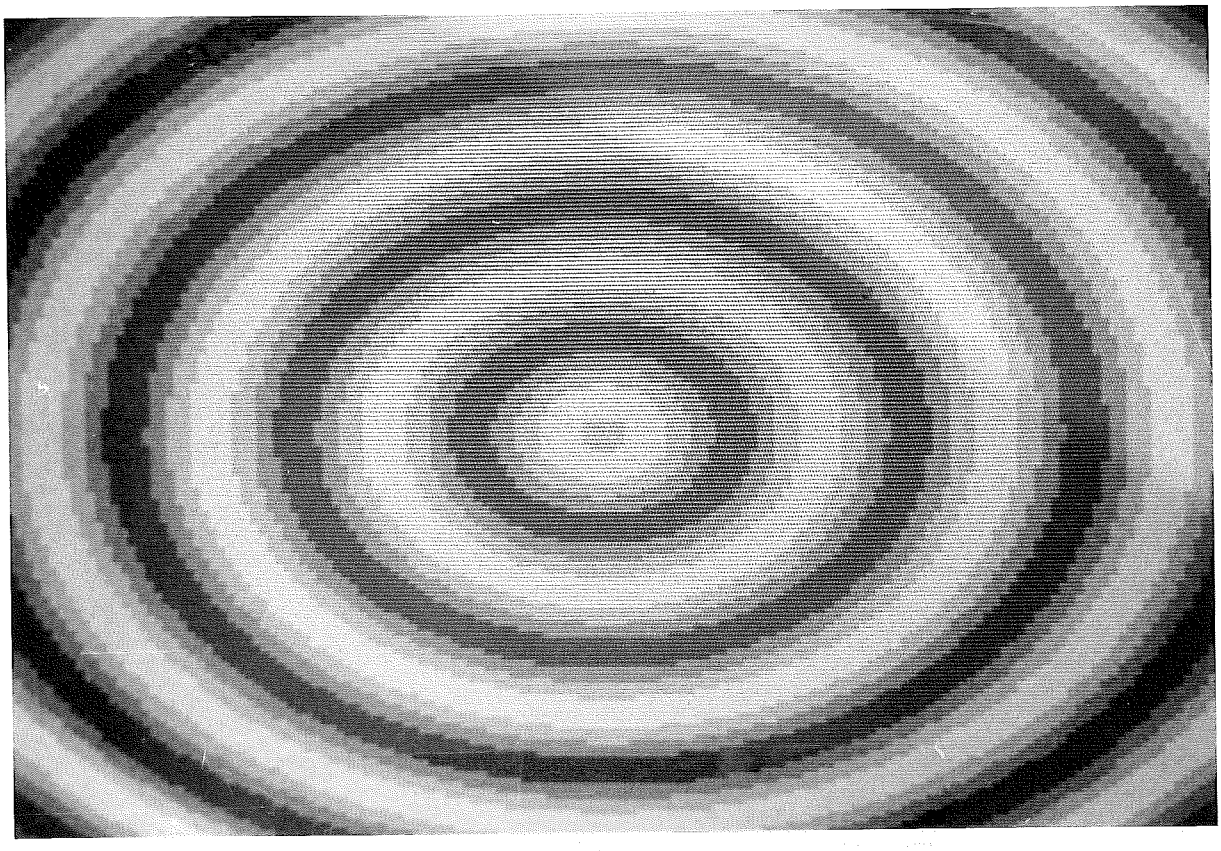


Image 7.1 C. M=170 N=170 b=2 M·N·b=57800

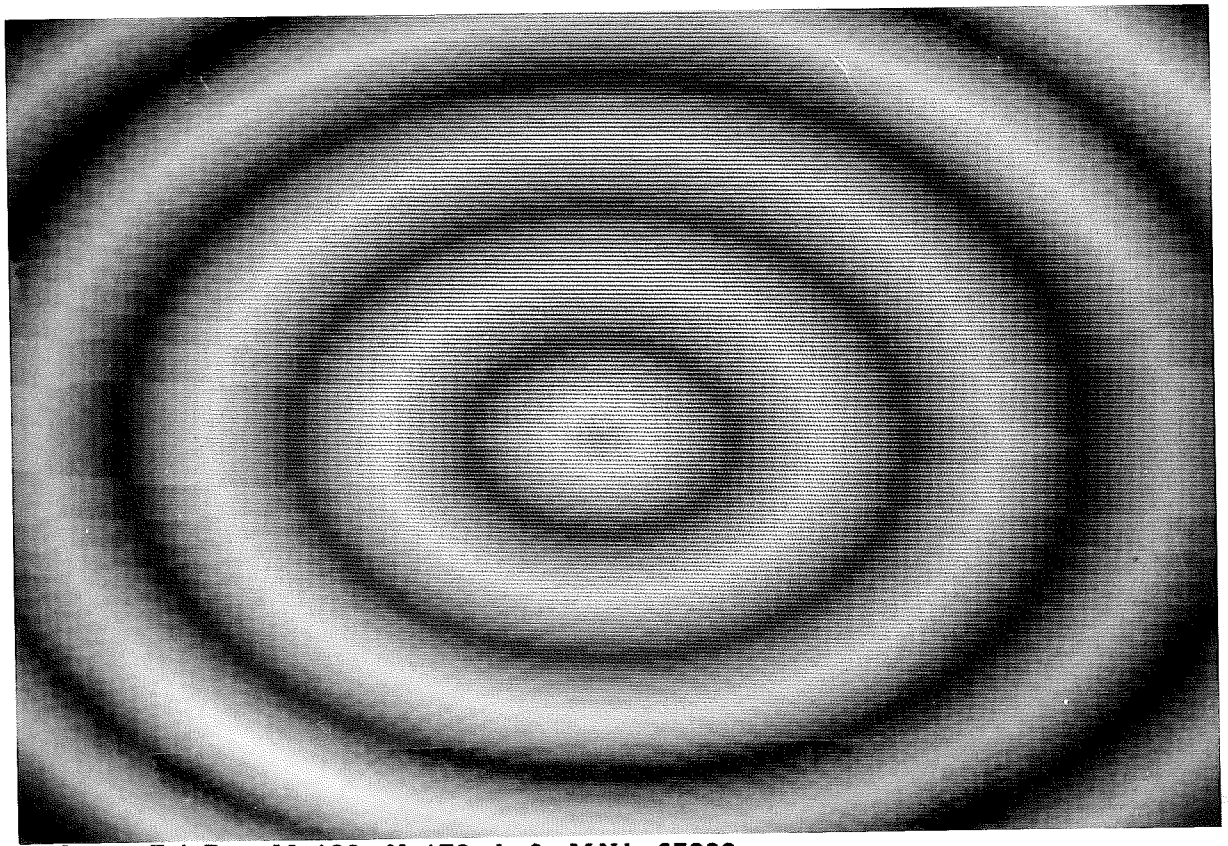


Image 7.1 D. M=128 N=170 b=3 M·N·b=65280

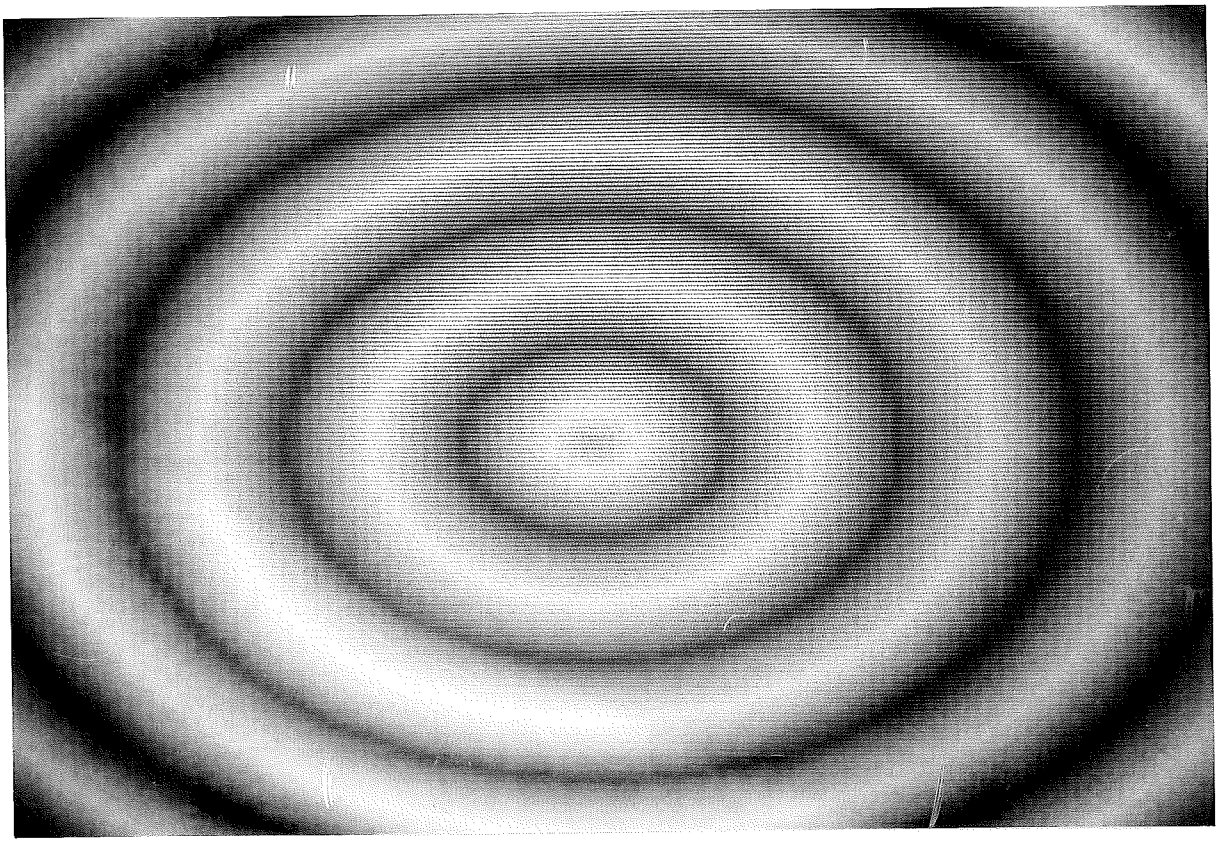


Image 7.1 E. M=128 N=128 b=4 M·N·b=65536

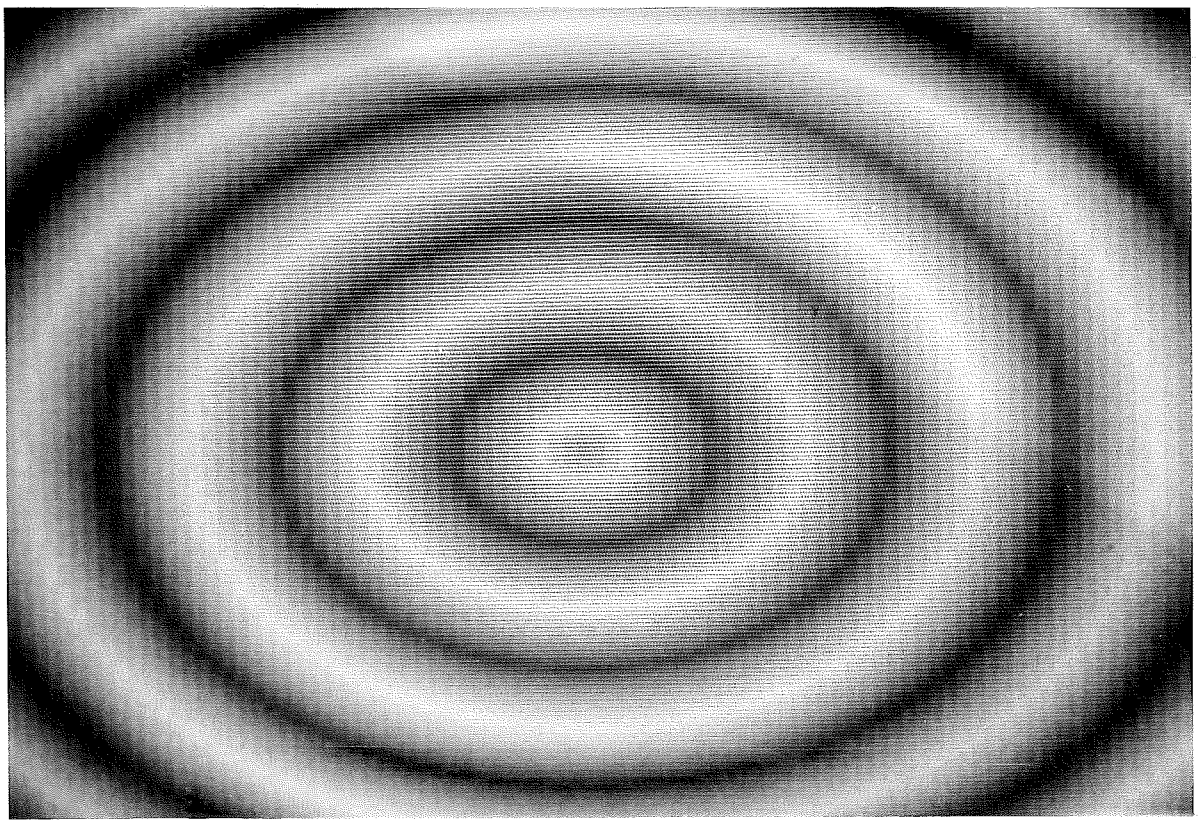


Image 7.1 F. M=102 N=128 b=5 M·N·b=65280

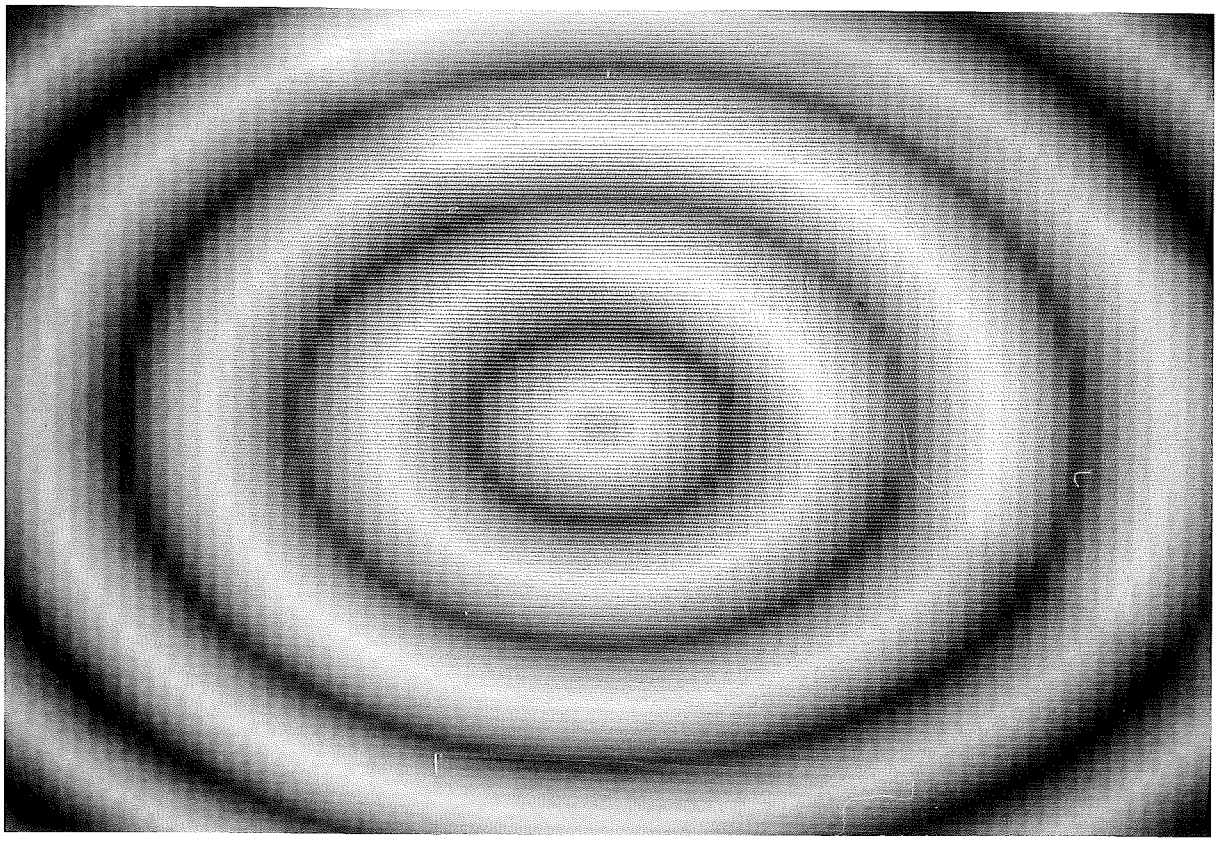


Image 7.1 G. M= 85 N=102 b=7 M·N·b=60690

Image 7.2 A. Original authors' image

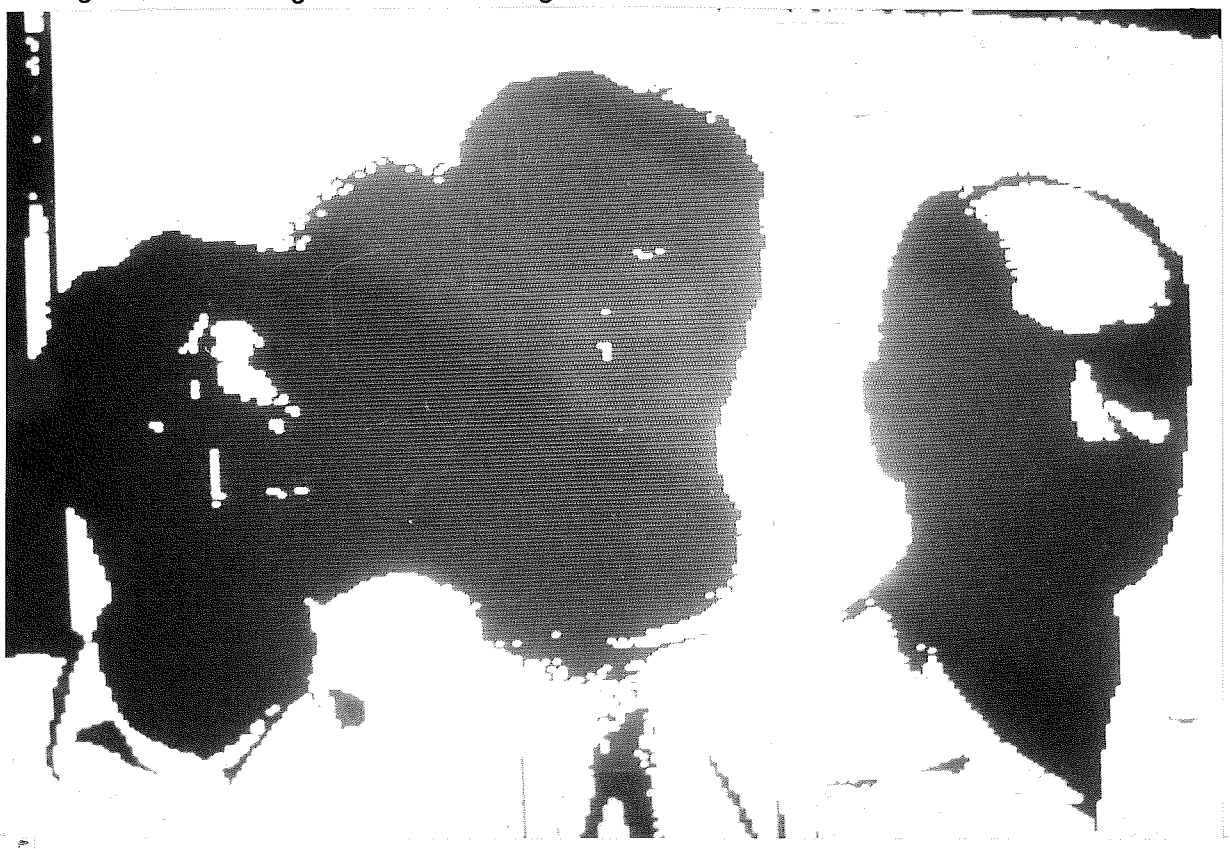


Image 7.2 B. M=256 N=256 b=1 M·N·b=65536

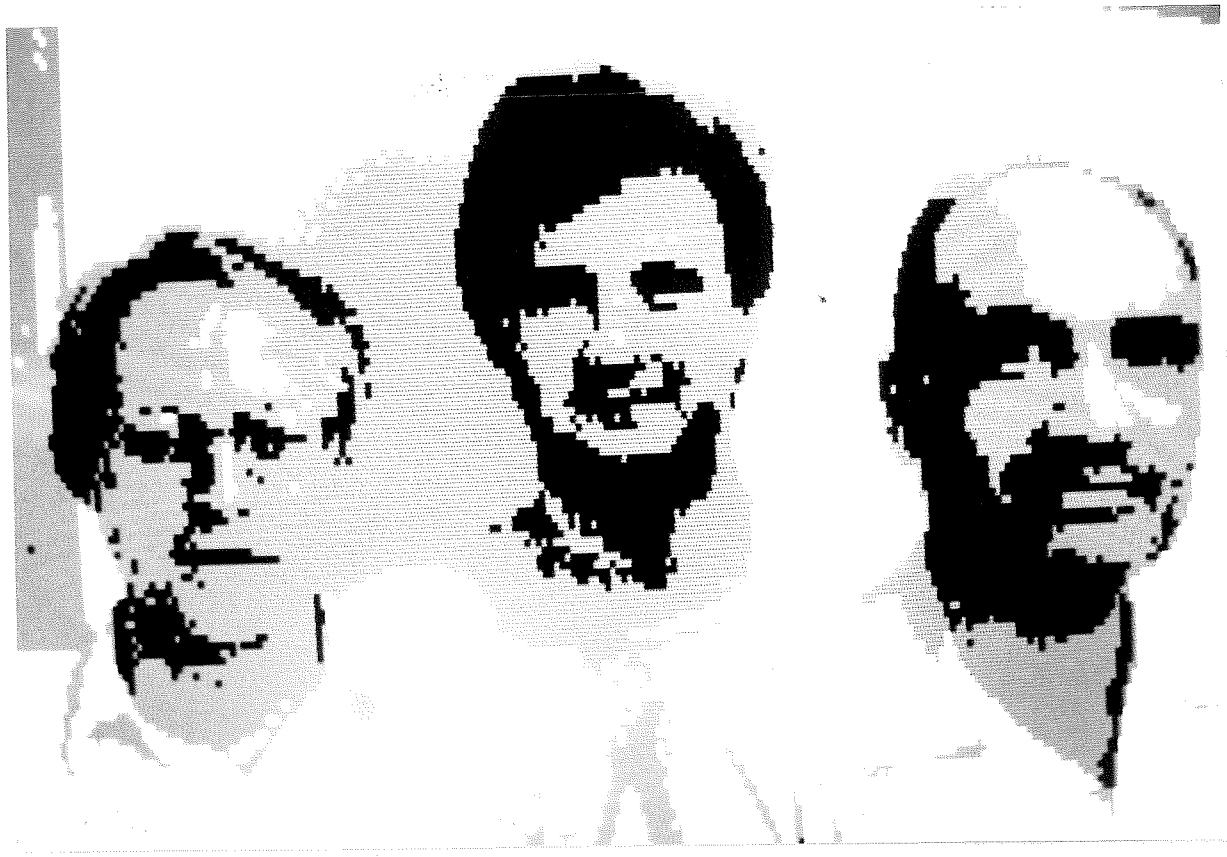


Image 7.2 C. M=256 N=128 b=2 M·N·b=65536

Image 7.2 D. M=170 N=102 b=3 M·N·b=52020

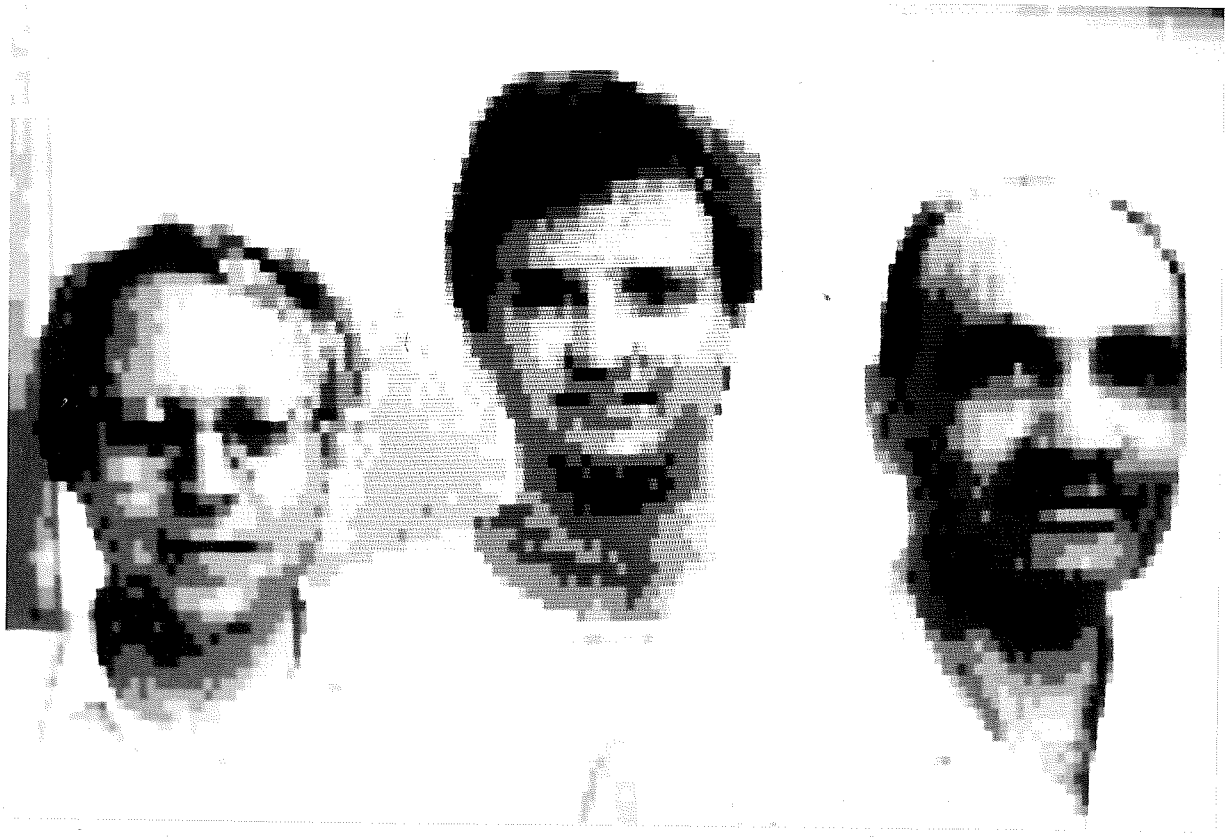


Image 7.2 E. M=170 N= 85 b=4 M·N·b=57800

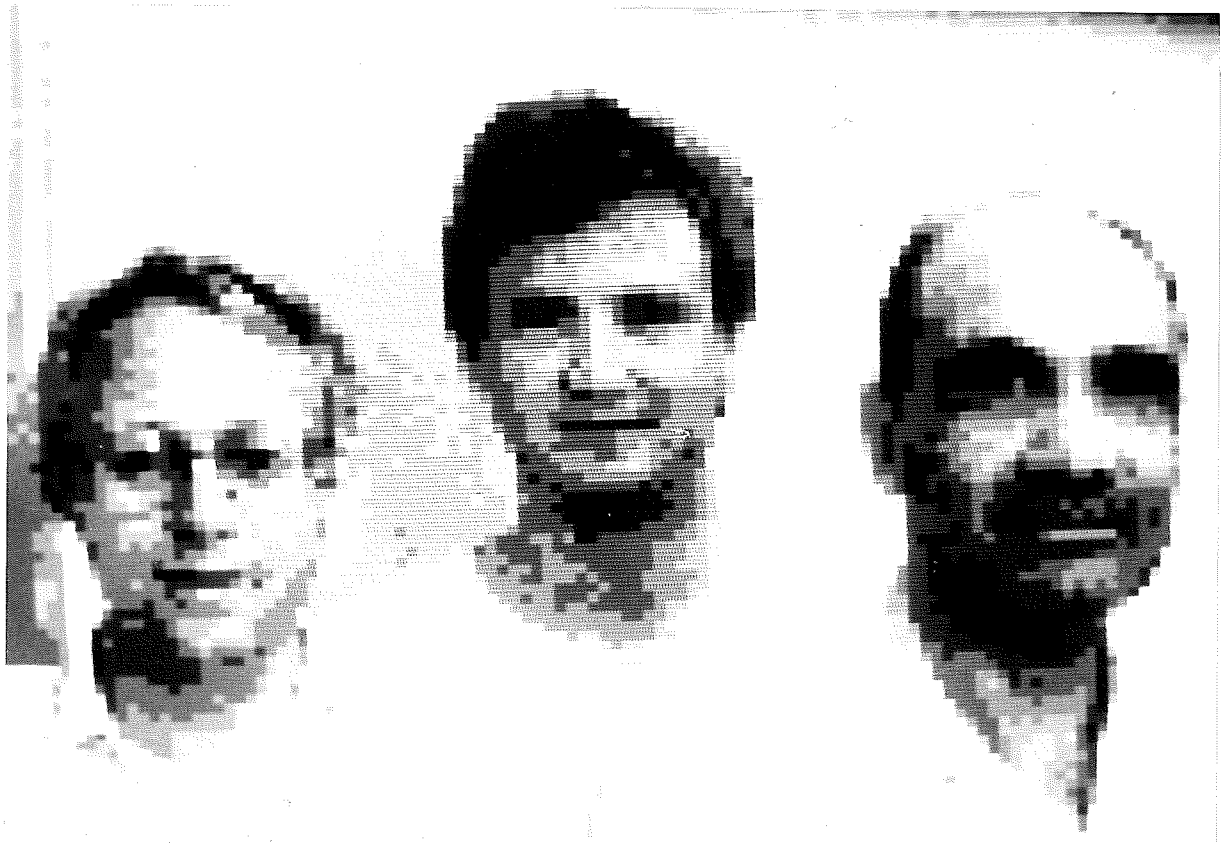


Image 7.2 F. M=128 N=102 b=4 M·N·b=52224

Image 7.2 G. M=170 N= 73 b=5 M·N·b=62050

Image 7.2 H. M=128 N= 64 b=7 M·N·b=57344

8. Conclusions

There is not yet a clean problem formulation of visual servoing which captures the essential properties and which admits an analytical or numerical solution. It is thus essential to find simplifications. This has been the theme of this thesis.

The use of sensory feedback, i.e. vision, touch, force etc., offers great possibilities for advanced automation. Chapter 3 indicates how it can be used in flexible storehouses for automatically guided vehicles with feedback from image and touch information. Sensory feedback is also of substantial interest in current robotics research. This research is, however, still at its infancy as is indicated by the following quote from Brady (1981, p. 11): "It would be a very brave person indeed who claimed to understand other than the broadest outlines of the subject now". This thesis is therefore based on an empirical approach, i.e. building a prototype system, making experiments, and trying to isolate parts of the problem for further theoretical and experimental investigations.

A visual servo laboratory was presented in Chapter 2. The conclusion is that it is now easy to develop a flexible system for study of solution principles from standard components. The visual servo example in Chapters 3 and 6 includes different aspects of visual servoing such as image interpretation, closed loop feedback, planning and the man-robot interface. A number of nontrivial decisions on selection of features, thresholds etc. are involved. Motion planning is used to simplify programming and interaction at the task level. Flexibility during unsupervised operation is achieved by repeating the planning when further information is obtained. Color graphics manipulation in real images of the robot working scene simplifies robot motion programming. The main idea is that the man-robot interface automatically translates the graphics entered by pointing in the image to process characteristics.

The two main theoretical contributions in this thesis are the study of the marking problem and the optimal digitization problem treated in Chapters 5 and 7 respectively. The formulation of the perspective invariant marking problem and its solution is new. The existence of perspective invariants based on areas is a key contribution. These invariants are shape dependent contrary to the case of parallel projection. The constructed symbols can be used for robot marking or as signposts. The optimal digitization gives the trade-off between sampling and quantization analytically. The principal behavior was verified experimentally. It also explains earlier subjective tests.

Future research is closely related to the dynamic hardware development. The image characteristics used in the Chapters 4-7 can be generated with readily available fast special hardware. In Chapter 4-6 each image is first treated to produce an edge-description. The image is then expressed as a collection of edges. In the future it may be possible that ideas on substantially more complex vision models have matured to implementation. The research in the field of artificial intelligence may offer such possibilities (Artificial Intelligence, 1981; Marr, 1982).

There are also interesting issues in the interaction between sensors and signal processing. A first straightforward example is automatic aperture control, which copes with varying light conditions. This has been used in this thesis. More complex situations arise when the camera and an image preprocessor is viewed as the sensor. The interaction with the interpretation stages can be based on knowledge and constraints in a "top-down" approach (Ballard and Brown, 1982). Further the knowledge of the response to the control signals can be used in the signal processing. The subimage selection in the visual servo example uses the predicted position and the predicted size of the object. Modern control theory in many cases results in considerably more complex internal structures of a servo (Åström and Wittenmark, 1984). The idea of observers and the idea of prediction of measurements (images) have been investigated in connection with missile guidance (Kaehr and Spector, 1980). These techniques may in the future be developed to cope with the scenes discussed in this thesis. The overall conclusion is that proper choices of vision models are nontrivial in visual servos and that it is not clear what the proper hardware should look like. The hope is that experiments in a slow time scale like in this thesis can be used to explore different ideas without being hampered by hardware limitations.

9. References

Abdou, I.E. and K.Y. Wong (1982). Analysis of linear interpolation schemes for bi-level image applications. IBM J. Res. Develop. vol. 26, no. 6, pp. 667-680.

Abelson, H. and A.A. diSessa (1981). Turtle Geometry. The Computer as a Medium for Exploring Mathematics. The MIT Press, Cambridge, Mass.

Andersson, R.L. (1984). A VLSI circuit for the real-time computation of moments. Preprint. NSF-STU International Workshop on Computer Vision and Industrial Applications, May 14-18, Stockholm, Sweden.

Artificial Intelligence (1981). Journal vol. 17, special issue on Vision, no:s 1-3.

Åström, K.J. (1975). A laboratory vehicle. Report CODEN: LUTFD2/(TFRT-3122), Department of Automatic Control, Lund Institute of Technology, Sweden.

Åström, K.J. and J.J. Anton (1984). Expert control. IFAC World Congress, also to appear in Automatica.

Åström, K.J. and B. Wittenmark (1984). Computer Controlled Systems - Theory and Design. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Ballard, D.H. and C.M. Brown (1982). Computer Vision. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Brady, M. (1981). Preface - The changing shape of computer vision. Artificial Intelligence, vol. 17, no:s 1-3, pp. 1-15.

Brady, M. and A. Yuille (1983). An extremum principle for shape from contour. A.I. Memo no. 711, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, Mass.

Brady, M., J.M. Hollerbach, T.L. Johnson, T. Lozano-Perez and M.T. Mason (Eds.) (1982). Robot Motion Planning and Control. The MIT Press, Cambridge, Mass.

Coxeter, H.S.M. (1961). Introduction to Geometry. John Wiley & Sons, Inc., New York.

Dahl, O. (1985). Master thesis to appear. Department of Automatic Control, Lund Institute of Technology, Sweden.

Elmqvist, H. (1983). Combining graphical descriptions and equations for dynamical modelling. Fourth IASTED International Symposium on Modelling and Simulation, June 21-24, Lugano, Switzerland.

Fogarty, J. (1969). Invariant Theory. W.A. Benjamin, Inc., New York.

Fu, K.S. (Ed.) (1977). Syntactic Pattern Recognition, Applications. Springer-Verlag, Berlin Heidelberg.

GPC 25 Black and White TV-Camera. Operating and Service Manual. 2K Video Systems, The Incentive Group.

Huang, T.S., O.J. Tretiak, B. Prasada, Y. Yamaguchi (1967). Design considerations in PCM transmission of low resolution monochrome still pictures. Proc. IEEE, vol. 55, pp. 331-335.

Jansson, S-O. (1985). Master thesis to appear. Department of Automatic Control, Lund Institute of Technology, Sweden.

Johansson, K. (1984). Ett exempel på robotpositionering med hjälp av videokamera. Master thesis CODEN: LUTFD2/(TFRT-5307), Department of Automatic Control, Lund Institute of Technology, Sweden.

Kaehr, R. and M. Spector (1980). USAF development of optical correlation missile guidance. Proc. Image Processing for Missile Guidance, July 29 - August 1, vol. 238, pp. 28-35.

Kay, A. and A. Goldberg (1977). Personal dynamic media. Computer, Micro-processors and Education, 10:1-6, pp. 31-41.

MACSYMA Reference Manual (1983). The Mathlab Group, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Mass.

Marr, D. (1982). Vision. W.H. Freeman & Co., San Francisco.

Matrox RGB-Graph Color Graphics Controller, Manual no. 167MO-04-0, Matrox Electronic Systems.

Matrox VAF-512 Graphics Support Board, Manual no. 176MO-01-0, Matrox Electronic Systems.

Moravec, H. (1977). Rover visual obstacle avoidance. 5th International Joint Conference on Artificial Intelligence, Massachusetts Institute of Technology, Cambridge, Mass.

Newman, W.M. and R. F. Sproull (1978). Principles of Interactive Computer Graphics. McGraw-Hill.

Nielsen, L. (1984a). Bildtolkning i ett exempel på robotpositionering med hjälp av videokamera. Symposium Bildanalys, Svenska Sällskapet för Automatiserad Bildanalys (SSAB), Linköping 15-16 mars.

Nielsen, L. (1984b). Motion detection in image sequences. Preprint, NSF-STU International Workshop on Computer Vision and Industrial Applications, May 14-18, Stockholm, Sweden.

Nielsen L. and H. Elmqvist (1983). An image laboratory. Report CODEN: LUTFD2/(TFRT-7261), Department of Automatic Control, Lund Institute of Technology, Sweden.

Nielsen, L. and K. Johansson (1984). Robot med egna ögon. Industriell Datateknik 1984:5, pp. 25-29.

Nielsen, L., K.J. Åström, E.I. Jury (1983). Optimal digitization of 2-D images. Report CODEN: LUTFD2/(TFRT-7265), Department of Automatic Control, Lund Institute of Technology, Sweden.

Nielsen, L., K.J. Åström, E.I. Jury (1984). Optimal digitization of 2-D images. IEEE Trans. on Acoustics, Speech, and Signal Processing, vol. 32, pp. 1247-1249.

Pavlidis, T. (1978). A review of algorithms for shape analysis. Computer Graphics and Image Processing, vol. 7, pp. 243-258.

Pavlidis, T. (1980). Algorithms for shape analysis of contours and waveforms. IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. PAMI-2, no. 4, pp. 301-312.

Pavlidis, T. (1982). Algorithms for Graphics and Image Processing. Springer-Verlag, Berlin-Heidelberg.

Petersson, C.U. (1983). An integrated robot vision system for industrial use. Rovisec-3, Third International Conference on Robot Vision and Sensory Control, Nov. 6-10, Cambridge, Mass.

Pratt, W.K. (1978). Digital Image Processing. John Wiley & Sons, Inc., New York.

Rosenfeld, A., A.C. Kak (1982). Digital Picture Processing. Academic Press, New York.

Sandin, G. and T. Wullt (1982). Experiment med datorstyrd kännande robot. Master thesis CODEN: LUTFD2/(TFRT-5271), Department of Automatic Control, Lund Institute of Technology, Sweden.

Silverberg, P., L. Nielsen, P. Omling, L. Samuelsson (1985). EL2-maps from computer based image analysis of semi-insulating GaAs wafers. Symposium on Defect Recognition and Image Processing in III-V Compounds, July 2-4, Montpellier, France.

Steiglitz, K. (1966). Transmission of an analog signal over a fixed bit-rate channel. IEEE Trans. on Information Theory. vol. IT-12, no. 4, pp. 469-474.

Terrapin Turtle Assembly Manual (1979). Instruction and Assembly Manual, Terrapin, Inc. 678, Massachusetts Avenue 205, Cambridge.

Williams, G. (1983). The Lisa computer system. BYTE Publication Inc., February, pp. 33-46.

Winston, P.H. (1977). Artificial Intelligence. Addison-Wesley Publ. Company.

A. Packages

The specification part of the basic support packages are listed. They are

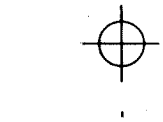
Raster.pak
RasterReg.pak
LookUp.pak
ImageDef.pak
ADConv.pak
DigConv.pak
SimplFil.pak

The package Raster is implemented in assembler. All other packages are implemented in Pascal.

```
------------------------------------------------------------

package RASTER is

-- Procedures for Raster Operations

-- Authors: Tommy Essebo, Lars Nielsen
-- Date: 1983-09-06

If the logical name NORAST is defined when MapRaster is
called the actual connection to the raster hardware is
inhibited.

If the logical name RASTCHECK is defined when MapRaster
is called a parameter check is made in various
procedure calls and a diagnostic printout occurs
if an error is found.

------------------------------------------------------------

.TYPE

bytes = 0..255;

.EXTERN

function MapRaster: integer; extern;
  { Initialization routine. Handles the mapping of registers
    into the virtual memory. A call to this procedure should
    be done first a user program. No user privileges are
    necessary. }

procedure MultiSet(nr: integer; word: integer); extern;
  { Write on Multibus address nr.  MultiSet and MultiGet are
    the fundamental operations to be able to read and write
    on any Multibus address. All other operations may be
    expressed in these, but for reasons of convenience or
    speed they are implemented as separate routines. }

procedure MultiGet(nr: integer; var word: integer); extern;
  { Read on Multibus address nr. }

procedure SetRasterReg(nr: integer; word: integer); extern;
  { Write on address nr in active plane. }

procedure GetRasterReg(nr: integer;var word: integer); extern;
  { Read on address nr in active plane. }

{------------------------------------------------------------}

procedure RasterRead(var image: array [n1..n2: integer] of
                packed array[m1..m2: integer] of bytes;
                xc, yc, nx, ny, increment: integer); extern;
  { Reads area (xc, yc) , (xc+nx, yc+ny) from Raster memory
    to the variable image.
    Note that (xc,yc) is upper left corner of the area and
    that (xc+nx,yc+ny) is lower right corner.
    (xc,yc) will be stored in image[ 1, 1].}

procedure RasterWrite(var image: array [n1..n2: integer] of
                packed array[m1..m2: integer] of bytes;
                xc, yc, nx, ny, increment: integer); extern;
```

```
  { Writes area (xc, yc) , (xc+nx, yc+ny) to Raster memory
    from the variable image.
    Note that (xc,yc) is upper left corner of the area and
    that (xc+nx,yc+ny) is lower right corner.
    (xc,yc) will be fetched from image[ 1, 1].}

procedure RasterMS4Read(var image: array [n1..n2: integer] of
                packed array[m1..m2: integer] of bytes;
                xc, yc, nx, ny, increment: integer); extern;
  { Same as RasterRead but only Most Significant 4 bits. }

procedure RasterMS4Write(var image: array [n1..n2: integer] of
                packed array[m1..m2: integer] of bytes;
                xc, yc, nx, ny, increment: integer); extern;
  { Same as RasterWrite but only Most Significant 4 bits. }

{-------------------------------------------------------------}

procedure VisiblePlane(nr: integer; visible: boolean); extern;
  { Set plane visibility status. }

procedure DrawPlane(nr: integer); extern;
  { Enables drawing in plane nr. }

procedure RasterErase(color: integer); extern;
  { Clears active plane to value color. }

procedure RasterCol(color: integer); extern;
  { Specify a new color. }

function ReadPixel(x, y: integer): integer; extern;
  { Get value at (x,y). }

procedure WritePixel(x, y: integer); extern;
  { Write a dot of current color in (x,y). }

{-------------------------------------------------------------}

procedure RasterClip(x1,y1,x2,y2: integer); extern;
  { Set limits for VerLine and HorLine. }

procedure RasterVerLine(x1,y1,y2: integer); extern;
  { Draw vertical line. }

procedure RasterHorLine(y1,x1,x2: integer); extern;
  { Draw horizontal line. }

procedure RasterPaint(xlow,ylow,xhigh,yhigh: integer); extern;
  { Paint a window on screen in current color. }

.END
```

```
------------------------------------------------------------

package RasterReg is

-- Procedures to initialize and handle the Matrox registers
-- Reference: Matrox VAF-512 manual page 24

-- Author: Lars Nielsen
-- Date: 1983-09-06

------------------------------------------------------------

.FORWARD

procedure GrabOne; forward;
  { Grab one frame and freeze it. }

procedure GrabCont; forward;
  { Do continuous frame grabbing to Raster memory. }

procedure Camera; forward;
  { Select the video camera as input. ( Default ) }

procedure Recorder; forward;
  { Select the video recorder as input. }

.END

------------------------------------------------------------

------------------------------------------------------------

package LookUp is

-- Procedures to program the look up table

-- Authors: Hilding Elmqvist, Lars Nielsen
-- Date: 1983-09-06

The look up table has 256 entries numbered 0 .. 255.
Each entry has an eight bit register for each of the
colors r, g and b.

------------------------------------------------------------

.FORWARD

procedure SetColorMap(color, r, g, b: integer); forward;
  { Set r g b values in entry color }

procedure GetColorMap(color: integer;var r,g,b: integer); forward;
  { Get r g b values from entry color }

procedure BlackWhite16; forward;
  { Initializes color look up table for 16 uniformly
    distributed intensities of black and white. }

procedure BlackWhite256; forward;
  { Initializes color look up table for 256 uniformly
    distributed intensities of black and white. }

procedure TwoPlanes; forward;
  { Initializes color look up table for
    two planes of 16 colors. }
```

```
procedure Pseudo; forward;
  { Initializes color look up table for 16 pseudo colors. }
procedure ColorPlane; forward;
  { Initializes color look up table for 256 pseudo colors. }
.END
```

```
package IMAGEDEF is
-- Definitions and procedures for image handling
-- Author: Tommy Essebo
-- Date: 1982-11-02
The raster memory is represented by a matrix of bytes:
image[row, column] , where 1 <= row, coloumn <= 512.
The top left pixel is image[1,1] and lower left corner
is in image[512, 1].
```

```
.CONST
imagesize = 512;
.TYPE
imageline = packed array [1..imagesize] of bytes;
imagetype = array [1..imagesize] of imageline;
.FORWARD
procedure ReadImage(var image: imagetype); forward;
  { Reads one image from Raster memory. }
procedure WriteImage(var image: imagetype); forward;
  { Writes one image to Raster memory. }
.END
```

```
package ADCONV is
-- Procedures for A/D and D/A conversion
-- Author: Tommy Essebo
-- Date: 1983-01-27
The analog values are real numbers in the range -1 to 1.
A/D input  channels: 0 - 15
D/A output channels: 0 - 5
AnalogOut will limit the values if needed.
```

```
.FORWARD
```

```
function AnalogIn(chan: integer): real; forward;
procedure AnalogOut(chan: integer; val: real); forward;
.END
-------------------------------------------------------------

-------------------------------------------------------------
package DIGCONV is
-- Procedures for digital input and output
-- Author: Tommy Essebo
-- Date: 1983-02-09
The digital values are boolean values.
Input  channels: 0 - 15
Output channels: 0 - 15
-------------------------------------------------------------
.FORWARD
function DigitalIn(chan: integer): boolean; forward;
procedure DigitalOut(chan: integer; val: boolean); forward;
.END
-------------------------------------------------------------

-------------------------------------------------------------
package SIMPLEFILER is
-- Procedures to save and restore imagefiles.
-- The image is 512x512 bytes.
-- Author: Lars Nielsen
-- Date: 1983-09-06
-------------------------------------------------------------
.VAR
imagefile                : file of imageline;
.FORWARD
procedure Restorefile(var image: imagetype); forward;
  { Asks for a filename and then reads the content of
    the file to the Pascal matrix image. }
procedure Savefile(var image: imagetype); forward;
  { Asks for a filename. Writes the Pascal matrix image
    to this file. }
.END
-------------------------------------------------------------
```

B. Turtle Control

Details of the algorithms of the visual servo will be given. The objective of the visual servo is to control the motion of the Turtle. It is viewed as an example of robot motion control or control of an automatically guided vehicle (AGV). The robot is known with respect to shape and dynamics. It is computer controlled and has touch sensors. Different Turtles are popular in the literature (Abelson and diSessa, 1980; Winston, 1977). This includes both mechanic and graphic Turtles. They have inspired Turtle graphics, Logo, and Turtle geometry.

It will be described how our Turtle is controlled to follow a path, using the primitive motion commands of Turtle graphics. Image feedback is used. The position of the marking symbol at the top of Turtle is determined in the image. This is transformed to the position on the floor. The video camera is placed to get an overview of the scene, and is fixed in position. It will also be described how the Turtle can follow an object contour by use of the touch sensors. The Turtle cannot move with its dome in constant contact with an object. Instead the touch sensing is based on repeated collisions like in Winston (1977, p. 248).

Primitive commands

The Turtle has two individually driven wheels. If both wheels move forward then the robot moves forward. If the right wheel moves forward and the left wheel moves backward then the robot rotates to the left around its center. A set of four motion procedures are used to command the robot motion:

```
Forw(length), Backw(length), TurnLeft(angle), TurnRight(angle)
```

These procedures are similar to the Turtle procedure notation (Abelson and diSessa, 1980, Appendix A). The length and angle are converted to the time the on/off signals of the Turtle motors have to be set to obtain a move of a certain distance or a rotation of a certain angle. The relations between set time and

distance respective angle are approximated to be linear. The conversion factors have been determined experimentally. We will here use

```
Turn(angle)
```

instead of TurnLeft(angle) or TurnRight(angle) to simplify the writing. Positive argument means left and negative means right.

These procedures are combined to a procedure for moving the Turtle to a desired position from its actual position. The state of the control is the last measured position and heading of the Turtle. The procedure is denoted

```
MoveTo(position)
```

and is performed by first turning the desired angle and thereafter moving forward the desired length i.e. as

```
Calculate(in position; out length,angle)
Turn(angle)
Forw(length)
```

The robot moves backwards only for fine adjustment of its position and after a collision. Before going into path following, contour following, and path finding we discuss what type of errors that is considered when choosing the Turtle control strategies.

Uncertainties

We distinguish between three types of uncertainties in the Turtle position. These errors are vectors but we will work only with the magnitude. The first type is static and is present even when the Turtle stands still. The top of the Turtle may vary the distance r forward or backward depending on if the Turtle leans on the forward support or on the backward support. We have here

$$r < 2 \text{ cm} \qquad \text{(B.1)}$$

The other two types of errors are motion disturbances. We differ between normal and large unnormal disturbances. The disturbances during normal robot motion are mainly due to the varying friction between the robot wheels and the floor and to backlash in the drive train. The backlash results in one wheel starting before the other. Furthermore the robot wiggles when moving forward. These errors result in motion errors as in Figure B.1. The error in position R after a MoveTo command has been determined experimentally. The main error

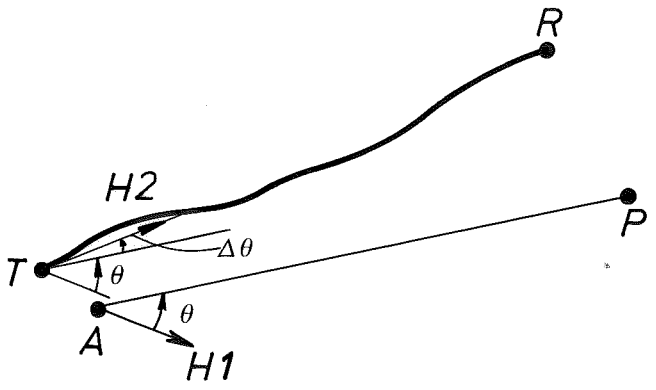


Figure B.1 Comparison between the intention and the result of a MoveTo(P) command. The denotations are: A - initial position; H1 - initial heading; P - desired predicted position after MoveTo(P); T - position after TurnLeft(θ); H2 - heading after TurnLeft(θ); Δθ - direction error obtained during TurnLeft(θ); G - resulting position.

source is the direction error Δθ obtained. This error is normally less than 10 degrees. The error R is the sum of r in (B.1) and the normal motion errors as in Figure B.1. A rough estimate that relates R to the length L in forw(L) of a MoveTo command is

$$R < \frac{1}{3} L \qquad \text{(B.2)}$$

The relation (B.2) is valid at least for moves shorter than 30 cm. The precision is thus poor, but the advantage of feedback is that crude systems can be used.

Large unnormal disturbances may occur if the robot touches an obstacle or one robot wheel slips on e.g. a piece of paper. In these cases the robot looses the direction of motion, which results in an unpredictable error in the position.

PathFollowing

In the visual servo the Turtle path is restricted to polygons. The path following algorithm is

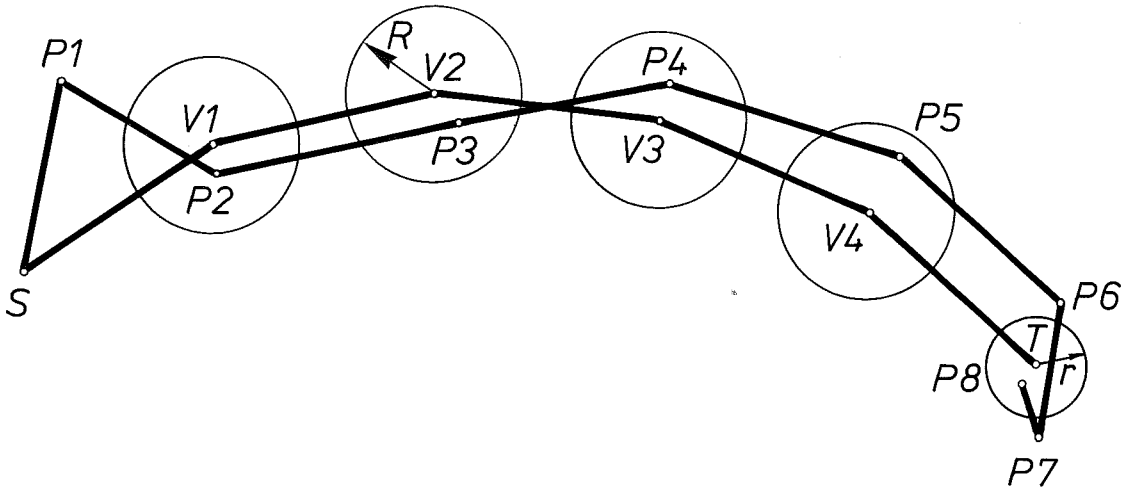


Figure B.2 An example illustrating the strategy for the path following. The path is described by S - starting point, T - terminal point, V1-V4 - points of the path.. A move is accepted (PositionReached is true in Figure B.2) if the robot is within the circles. The radius R for the circles around V1-V4 is chosen to be the upper bound for a normal motion error. If the robot is outside a circle then a large error has occured and a new MoveTo command to the same point is given. The radius r around T is based on the precision wanted. The points P1-P8 show an example of a motion.

```
  state = robot position and heading
  path = list of positions on the floor

PathFollowing is
  repeat
    position := pop(path)
    repeat
      MoveTo(position)
      Measure(position)
      update(state)
    until PositionReached
  until EndOfPath
```

Measure(position) is performed by the image processing, see Section 6.5. Each segment of the polygon is restricted to a maximal length of 15 cm, which from (B.2) gives the maximal error in resulting position after each MoveTo command as

$$R < 5 \text{ cm} \qquad \text{(B.3)}$$

The strategy for following of the path is illustrated in Figure B.2. A move is accepted if it is within the limits of normal motion disturbances (B.3). The first move in Figure B.2 resulted in P1 and a new move towards V1 is performed resulting in P2. At the terminal point we require the precision of (B.1). In the

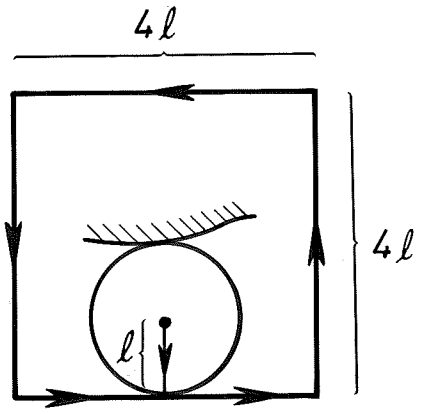


Figure B.3 TouchMotion. The search direction is here to the right (counter clockwise).

fine adjustment of the position the Turtle may move backward. This is the case between P7 and P8. Image 36 of Figure 3.4 gives another example of path following.

ContourFollowingByTouch

The contour following algorithm is based on repeated collisions. A basic touch motion TouchMotion is repeated. The Turtle first moves the distance ℓ away from the obstacle. Thereafter the path is a square with side 4ℓ. The path is followed clockwise or counter clockwise depending on the selection of the search direction described later. The case of counter clockwise search is illustrated in Figure B.3. We use ℓ = 15 cm. In the case of counter clockwise search then TouchMotion goes as follows

```
TouchMotion is
  Backw(2)
  Measure(position)
  Backw(ℓ-2)
  Measure(position)
  Turn(-90)
  Forw(2ℓ)
  repeat 4 times
    Turn(90)
    Forw(4ℓ)
```

The dome of Turtle may be tilted at a collision. Because of that it is first backed 2 cm to get free of the obstacle. Otherwise errors could be obtained in Measure(position). The position is measured again after Backw(ℓ-2) to verify in

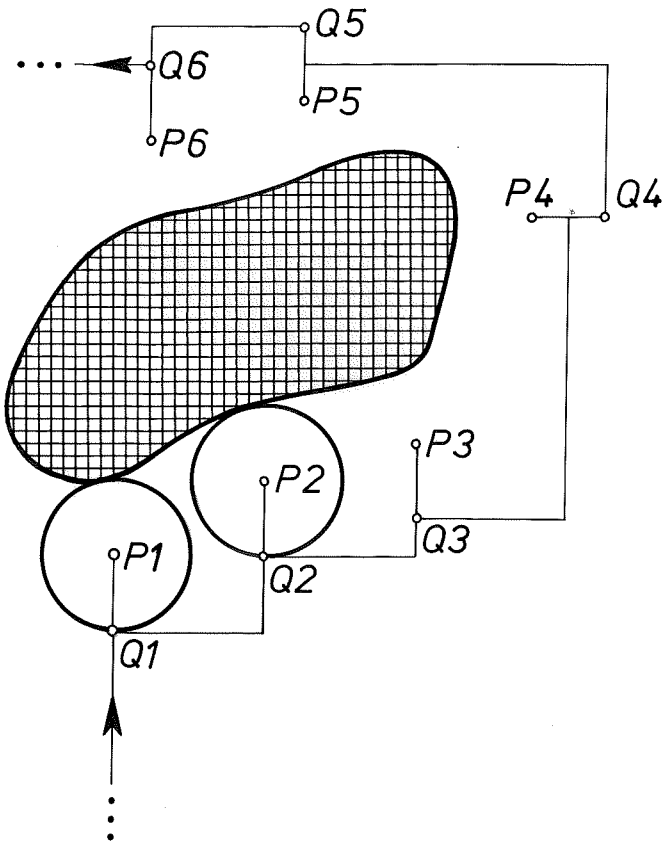


Figure B.4 An example of contour following. The points Pi are the Turtle positions at the obstacle boundary after a collision. The points represent the center and are hence one Turtle radius (+2 cm) away from the obstacle. The points Qi are the position after Backw(ℓ-2) in the TouchMotion.

what direction the obstacle is.

The basic search motion is repeated until a stop condition is fulfilled.

```
ContourFollowingByTouch is
  repeat
    TouchMotion
  until Finished
```

The Turtle will, while following the square path, collide again with the obstacle. An example of a result of the contour following is seen in Figure B.4. There are two cases for the criterion Finished. In the first case the contour following is continued until the object is encircled. In the second case the contour following is continued until the robot reaches the planned path on the other side of the object.

Contour description of obstacles

A contour description can be generated during the contour following by touch. The knowledge of the object contour is the points P_i and Q_i obtained from the position determination in TouchMotion during the contour following, see Figure B.4. The points P_i are one Turtle radius away from the object. The set of points $\{P_i\}$ have hence to be shrinked to form the geometric description of the object. In the case where the Turtle has encircled the object, a closed polygon is obtained. Each line segment is parallelly moved the distance of one Turtle radius to shrink the polygon. The other case is when the search stops when the path is reached. Then only an open part of the object contour is obtained as shown in Figure B.5, where a planned path have been added to Figure B.4. The contour segment is extended to a closed curve, by introducing two extra points in the curve segment. These points E1 and E2 are the distance 2ℓ behind the endpoints of the contour segment P1 and PN. In Figure B.5 we have N=4. By being behind a collision point is meant the direction defined by P1, Q1 and PN, QN respectively. The polygon shrinking is now performed only on the line segments corresponding to the obstacle contour.

SelectionOfSearchDirection

As described in Section 3.4 the Turtle has after a collision to decide on starting the contour following to the left or to the right. The algorithm for selection of search direction uses the obstacle map, the starting point S and the terminal point T of the path, the direction of the path at the collision point, and the direction from the collision point to the terminal point. A number of tests are performed. If the Turtle is on a path which is not the line ST, the reason is probably that the path is planned to avoid an obstacle. It is tested by using the obstacle map if the area between the rest of the path and the line ST is free of known obstacles. The tested area is determined as in Figure B.6. If the tested area is not free the Turtle selects the search direction to be away from the line ST. If the tested area is free then the direction of the obstacle contour is estimated. The Turtle moves in the direction leading toward the line ST and obtains a new collision point. The two collision points are projected onto the path and it is determined if the Turtle has moved forward or backward in the path direction. The search direction is chosen to be the direction in which the Turtle moves forward. To conclude: The search direction is chosen to be away from known obstacles. If there are no

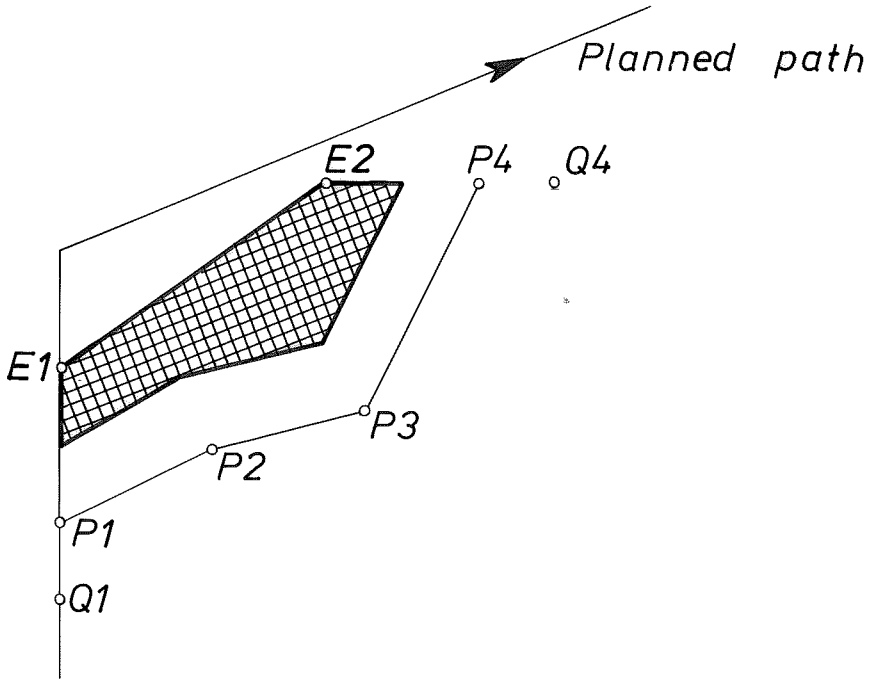


Figure B.5 The resulting object description obtained by the contour following algorithm. The object contour has been followed until the path is reached. The contour segment P1, P2,... is extended with the points E1 and E2. The polygon shrinking is performed only on the line segments corresponding to the obstacle contour.

known obstacles influencing the path, then the search direction is chosen to be the direction in which the robot moves forward.

```
SelectionOfSearchDirection is
  if CollisionPointRightOfLineST then
    AwayFromLineST := Right
  else AwayFromLineST := Left
  if TestArea not free then
    SearchDirection := AwayFromLineST
  else
    begin
    SearchDirection := not AwayFromLineST
    TouchMotion
    if "P2 < P1" then SearchDirection := not SearchDirection
    end
```

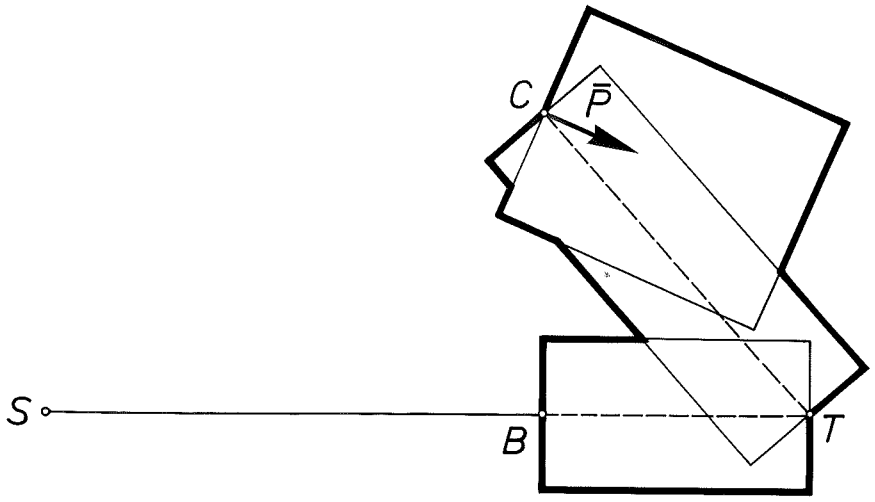


Figure B.6 The tested area in the algorithm for selection of search direction after a collision. The denotations are: S: the starting point of the path, T: the terminal point of the path, C: the collision point, B: the projection of C on the line between S and T, $\bar{P}$: the direction of the path at the collision point.
The tested area is a union of three rectangles. Two rectangles with the width of 2 robot diameters. One rectangle from C to T, and one rectangle from B to T. The third rectangle has the width of 3 robot diameters and goes from C the length Y in direction $\bar{P}$.

PathFinding

A very simple approach is taken for the path finding problem. The rationales and principles were given in Section 3.3.

The algorithm consists of two steps. First a path described by a two vertex polygon is searched. Then it is tried to smoothen the path using the two vertices as control points for a spline. The position of the two vertices V1 and V2 are varied in a simple search, and after each modification of the path it is tested with the use of the obstacle map if the path is avoiding the known obstacles. The scheme for variation of the path is illustrated in figure B.7. First the straight path between the starting point and the terminal point is tested. If the path is not free then a path segment V1 and V2 parallel to the straight path is tested. The path segment is moved alterly further away to each side. When a free path segment is found $V1_3$ and $V2_3$ then the straight paths from the segment terminal points to respectively the starting point S and $V1_3$ and terminal point $V2_3$ and T are tested. If either path is not free then the path segment is extended, see $V2_3$ and $V2_4$ in Figure B.7. If the path segment extension does not result in a free

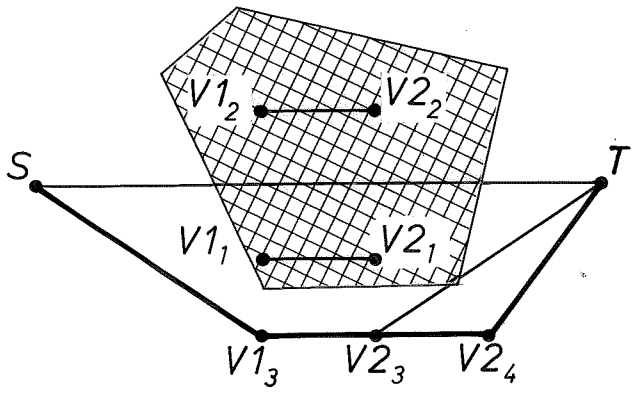


Figure B.7 An example of the search for an obstacle free way in the path finding algorithm. The denotations are: S: starting point of the path, T: terminal point of the path, $V1_i$, $V2_i$: vertices of the polygon path. The index i denotes the search order. The obstacle map contains one obstacle, which is situated between S and T. The search has gone as:

Path (S,T) not free → Path ($V1_1$, $V2_1$) not free → Path ($V1_2$, $V2_2$) not free →

Path ($V1_3$, $V2_3$) free

Path (S, $V1_3$) free Path ($V2_3$, T) not free →

Path ($V2_4$, T) free

which results in a free way from S to T.

path then the search is resumed by going further away with the parallel path segment.

When a free path consisting of a polygon with two vertices has been found then the program attempts to smoothen the path. The technique is described in Section 3.5. Briefly it consists of the calculation of a spline and then sampling it to a many vertex polygon. It is checked if the resulting path is free. If not then it is searched for a new two-vertex polygon. This is repeated. If a smoothed path is not found then the first free two-vertex polygon will be used. The approach works in several simple cases. It may be worth mentioning that in a first try to find an algorithm a one vertex polygon was used as a path, but the problem was to find paths that stayed within the image.

C. MACSYMA Runs

The investigations of this thesis have to a large extent been guided by the use of computer algebra. The program MACSYMA for symbolic manipulation of algebraic expressions has been used (MACSYMA, 1983). A number of MACSYMA programs (batch files) have been developed for perspective transformation. An object can be arbitrarily positioned in three-dimensional space, and the image of the object can then be computed. An object is a list of primitive objects, which in turn are lists of points in an object coordinate system. The derivation of (5.32) and a direct proof of the invariance of (5.54) will be given as an illustration.

Comments to the listing

The user commands are preceded by (c..) and the output from MACSYMA are preceded by (d..). If a command is concluded with an ; then the result is printed, but if it is concluded with $ the result is not printed. A comment is given as /* comment */.

The command (c3) defines an object (5.20), m=3 and n=4, as the variable Object. The variable PrimObject corresponds to $r_o(1)$, $r_o(2)$, and $r_o(3)$. The command (c12) computes the image of the object. The variable ObjectInSpace is R_c (5.11) and the variable Image is R_i (5.13). The areas in the image (5.25) are computed in (c28). The result is a list of the four areas and they are separated in (c30), (c31), (c32), and (c33). The result (5.32) in (d35) is derived by simplifying a2/a1. The invariance of (5.54) is proven by forming the invariant and then simplifying. The result (d44) is independent of x, y, z, theta, phi, and psi. The calculations used 4 hours and 40 minutes CPU time on the Vax.

```
(d1)                                appendix

(c2) batch(appc)$

(c3) batch(triangles)$

(c4) /* Define object consisting of 4 concentric equilateral triangles */

p1:matrix([b],[0],[0]);

                                    [ b ]
                                    [   ]
(d4)                                [ 0 ]
                                    [   ]
                                    [ 0 ]

(c5) p2:matrix([b*cos(2*%pi/3)],[b*sin(2*%pi/3)],[0]);

                                    [     b    ]
                                    [   - -    ]
                                    [     2    ]
                                    [          ]
(d5)                                [ sqrt(3) b ]
                                    [ --------- ]
                                    [     2    ]
                                    [          ]
                                    [     0    ]

(c6) p3:matrix([b*cos(4*%pi/3)],[b*sin(4*%pi/3)],[0]);

                                    [     b    ]
                                    [   - -    ]
                                    [     2    ]
                                    [          ]
(d6)                                [  sqrt(3) b ]
                                    [ - --------- ]
```


```
                          [      2     ]
                          [            ]
                          [      0     ]
```

```
(c7) PrimObject:[p1,p2,p3];
```

```
                                [      b     ] [      b     ]
                                [    - -     ] [    - -     ]
                          [ b ] [      2     ] [      2     ]
                          [   ] [            ] [            ]
(d7)                     [[ 0 ], [ sqrt(3) b ], [ sqrt(3) b ]]
                          [   ] [ ---------- ] [ - ---------- ]
                          [ 0 ] [      2     ] [      2     ]
                                [            ] [            ]
                                [      0     ] [      0     ]
```

```
(c8) PrimObjectSize:length(PrimObject);
```

```
(d8)                                3
```

```
(c9) Object:[k1*PrimObject,k2*PrimObject,k3*PrimObject,k4*PrimObject]$
```

```
(c10) ObjectSize:length(Object);
```

```
(d10)                               4
```

```
(c12) batch(imaging)$

(c13) /* Define global constants */

batch(rotate)$

(c14) rotatexaxis(angle):=
                      matrix(
                            [1,0,0],
```

```
                        [0,cos(angle),-sin(angle)],
                        [0,sin(angle),cos(angle)])$

(c15) rotateyaxis(angle):=
                      matrix(
                        [cos(angle),0,-sin(angle)],
                        [0,1,0],
                        [sin(angle),0,cos(angle)])$

(c16) rotatezaxis(angle):=
                      matrix(
                        [cos(angle),-sin(angle),0],
                        [sin(angle),cos(angle),0],
                        [0,0,1])$

(c17) ojler(theta,phi,psi):= rotatezaxis(theta).
                      rotatexaxis(phi).
                      rotatezaxis(psi)$


(c19) /* Euler rotation matrix */

Q: ojler(theta,phi,psi);

                [ cos(psi) cos(theta) - cos(phi) sin(psi) sin(theta) ]
                [                                                    ]
(d19)  Col 1 =  [ cos(psi) sin(theta) + cos(phi) sin(psi) cos(theta) ]
                [                                                    ]
                [                sin(phi) sin(psi)                   ]

          [ - cos(phi) cos(psi) sin(theta) - sin(psi) cos(theta) ]
          [                                                      ]
 Col 2 =  [  cos(phi) cos(psi) cos(theta) - sin(psi) sin(theta)  ]
          [                                                      ]
          [                sin(phi) cos(psi)                     ]
```


```
        [  sin(phi) sin(theta)  ]
        [                       ]
Col 3 = [ - sin(phi) cos(theta) ]
        [                       ]
        [       cos(phi)        ]
```

```
(c20) /* Translation vector */

r:matrix([x],[y],[z]);

                                  [ x ]
                                  [   ]
(d20)                             [ y ]
                                  [   ]
                                  [ z ]

(c21) /* Focal length */

f;

(d21)                               f

(c22) /* Transforms */

/* Positioning */

f(p):=r+Q.p;

(d22)                          f(p) := r + q . p

(c23) ObjectInSpace:makelist(map(f,Object[i]),i,1,ObjectSize)$

//macsyma/das/mstuff.o being loaded.

(c24) /* Imaging transformation */
```


```
t(p):=matrix([f*p[1]/(f+p[3])],[f*p[2]/(f+p[3])]);
```

$$(d24)\qquad t(p) := \mathrm{matrix}\left(\left[\frac{f\,p_1}{f+p_3}\right], \left[\frac{f\,p_2}{f+p_3}\right]\right)$$

```
(c25) Image:makelist(map(t,ObjectInSpace[i]),i,1,ObjectSize)$

(c27) batch(areas)$

(c28) Areas:makelist(
      (sum(determinant(addcol(image[j][i],image[j][i+1])),i,1,PrimObjectSize-1)
       +determinant(addcol(image[j][PrimObjectSize],image[j][1])))/2,
      j,1,ObjectSize)$

(c30) /* The area of the smallest triangle */

a1:part(areas,1,1);
```

$$(d30)\ \left(f^2\,(x + b\,k1\,(\cos(psi)\cos(theta) - \cos(phi)\sin(psi)\sin(theta)))\right.$$

$$\left(y + \frac{\sqrt{3}\,b\,k1\,(\cos(phi)\cos(psi)\cos(theta) - \sin(psi)\sin(theta))}{2}\right.$$

$$\left.\left. - \frac{b\,k1\,(\cos(psi)\sin(theta) + \cos(phi)\sin(psi)\cos(theta))}{2}\right)\right.$$

$$\left/\left(\left(z - \frac{b\,k1\,\sin(phi)\sin(psi)}{2} + \frac{\sqrt{3}\,b\,k1\,\sin(phi)\cos(psi)}{2} + f\right)\right.\right.$$

```
                                     2
  (z + b k1 sin(phi) sin(psi) + f)) - f

       b k1 (cos(psi) cos(theta) - cos(phi) sin(psi) sin(theta))
  (x - ----------------------------------------------------------
                                  2

    sqrt(3) b k1 (- cos(phi) cos(psi) sin(theta) - sin(psi) cos(theta))
  + --------------------------------------------------------------------)
                                     2

  (y + b k1 (cos(psi) sin(theta) + cos(phi) sin(psi) cos(theta)))

        b k1 sin(phi) sin(psi)    sqrt(3) b k1 sin(phi) cos(psi)
 /((z - ---------------------- + ------------------------------ + f)
                  2                             2

                                     2
  (z + b k1 sin(phi) sin(psi) + f)) - f

  (x + b k1 (cos(psi) cos(theta) - cos(phi) sin(psi) sin(theta)))

       sqrt(3) b k1 (cos(phi) cos(psi) cos(theta) - sin(psi) sin(theta))
  (y - -----------------------------------------------------------------
                                     2

    b k1 (cos(psi) sin(theta) + cos(phi) sin(psi) cos(theta))
  - ---------------------------------------------------------)
                              2

        b k1 sin(phi) sin(psi)    sqrt(3) b k1 sin(phi) cos(psi)
 /((z - ---------------------- - ------------------------------ + f)
                  2                             2

                                     2
  (z + b k1 sin(phi) sin(psi) + f)) + f
```

10 D H

```
     b k1 (cos(psi) cos(theta) - cos(phi) sin(psi) sin(theta))
(x - ---------------------------------------------------------
                                2

  sqrt(3) b k1 (- cos(phi) cos(psi) sin(theta) - sin(psi) cos(theta))
- ------------------------------------------------------------------)
                                2

(y + b k1 (cos(psi) sin(theta) + cos(phi) sin(psi) cos(theta)))

      b k1 sin(phi) sin(psi)   sqrt(3) b k1 sin(phi) cos(psi)
/((z - ---------------------- - ------------------------------ + f)
                2                          2

                                2
(z + b k1 sin(phi) sin(psi) + f)) - f

     b k1 (cos(psi) cos(theta) - cos(phi) sin(psi) sin(theta))
(x - ---------------------------------------------------------
                                2

  sqrt(3) b k1 (- cos(phi) cos(psi) sin(theta) - sin(psi) cos(theta))
- ------------------------------------------------------------------)
                                2

     sqrt(3) b k1 (cos(phi) cos(psi) cos(theta) - sin(psi) sin(theta))
(y + ---------------------------------------------------------------
                                2

  b k1 (cos(psi) sin(theta) + cos(phi) sin(psi) cos(theta))
- ---------------------------------------------------------)
                                2

      b k1 sin(phi) sin(psi)   sqrt(3) b k1 sin(phi) cos(psi)
/((z - ---------------------- - ------------------------------ + f)
                2                          2
```

```
     b k1 sin(phi) sin(psi)    sqrt(3) b k1 sin(phi) cos(psi)
(z - ---------------------- + ------------------------------ + f))
               2                          2

   2      b k1 (cos(psi) cos(theta) - cos(phi) sin(psi) sin(theta))
+ f  (x - ---------------------------------------------------------
                                  2

  sqrt(3) b k1 (- cos(phi) cos(psi) sin(theta) - sin(psi) cos(theta))
+ ------------------------------------------------------------------)
                                  2

     sqrt(3) b k1 (cos(phi) cos(psi) cos(theta) - sin(psi) sin(theta))
(y - ------------------------------------------------------------------
                                  2

  b k1 (cos(psi) sin(theta) + cos(phi) sin(psi) cos(theta))
- ---------------------------------------------------------)
                                  2

      b k1 sin(phi) sin(psi)    sqrt(3) b k1 sin(phi) cos(psi)
/((z - ---------------------- - ------------------------------ + f)
                2                          2

     b k1 sin(phi) sin(psi)    sqrt(3) b k1 sin(phi) cos(psi)
(z - ---------------------- + ------------------------------ + f)))/2
               2                          2
```

```
(c31) a2: part(areas,2,1)$

(c32) a3: part(areas,3,1)$

(c33) a4: part(areas,4,1)$
```

```
(c34) /* Derivation of expression 5.32 */

trigsimp(ratsimp(k1^2/k2^2*a2/a1));

//macsyma/share/trgsmp.l being loaded.
[fasl hole filled up]

          3         2         2      2  2    2
(d34) (4 z  + 12 f z  + (12 f  - 3 b  k1  sin (phi)) z

     3  3    3          3           3  3    3
 + 4 b  k1  sin (phi) sin (psi) - 3 b  k1  sin (phi) sin(psi)

     2    2    2           3     3         2
 - 3 b  f k1  sin (phi) + 4 f )/(4 z  + 12 f z

       2      2  2    2              3  3    3         3
 + (12 f  - 3 b  k2  sin (phi)) z + 4 b  k2  sin (phi) sin (psi)

     3  3    3                    2    2    2           3
 - 3 b  k2  sin (phi) sin(psi) - 3 b  f k2  sin (phi) + 4 f )

(c35) k2^2/k1^2*subst(z+f,help,ratsubst(help,z+f,trigreduce(%,psi)));

(d35)

   2            3      2  2    2                3  3    3
  k2  (- 4 (z + f)  + 3 b  k1  sin (phi) (z + f) + b  k1  sin (phi) sin(3 psi))
  ------------------------------------------------------------------------------
   2             3      2  2    2                3  3    3
  k1  (- 4 (z + f)  + 3 b  k2  sin (phi) (z + f) + b  k2  sin (phi) sin(3 psi))
```

```
(c36) /* A direct verification of the invariance of expression 5.54 */

batch(invariant)$

(c37) h(x1,x2,x3):=x1^2*x2^m-x1^m*x2^2
  +x2^2*x3^m-x2^m*x3^2+x3^2*x1^m-x3^m*x1^2;

                          2  m     m  2     2  m     m  2     2  m
(d37) h(x1, x2, x3) := x1  x2  - x1  x2  + x2  x3  - x2  x3  + x3  x1

                                                                  m  2
                                                               - x3  x1

(c38) l1:-k1^2*h(k2,k3,k4);

          2   2  m     2  m     m  2     m  2     2  m     m  2
(d38)  - k1  (k3  k4  - k2  k4  - k3  k4  + k2  k4  + k2  k3  - k2  k3 )

(c39) l2:k2^2*h(k3,k4,k1)$

(c40) l3:-k3^2*h(k4,k1,k2)$

(c41) l4:k4^2*h(k1,k2,k3)$

(c42) invariant:l1/a1+l2/a2+l3/a3+l4/a4$


(c44) ratsimp(ev(invariant,m:3));

(d44)                                  0

(c45) /*          Q.E.D.          */
```