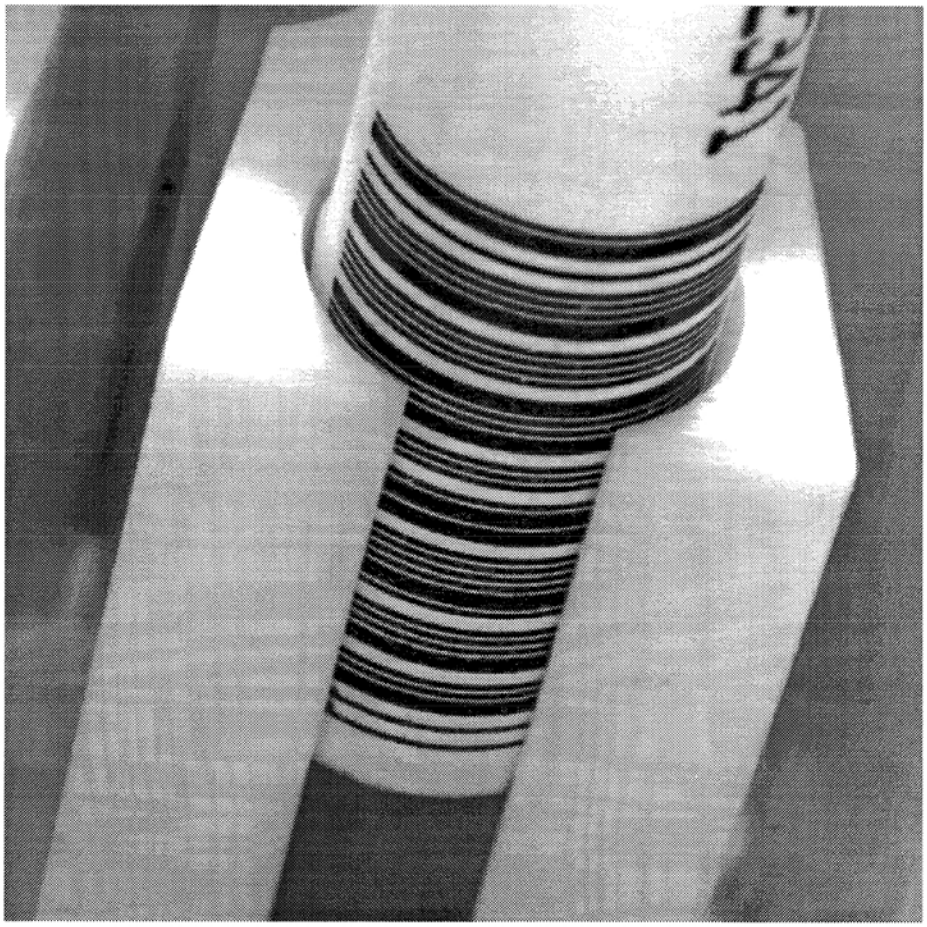
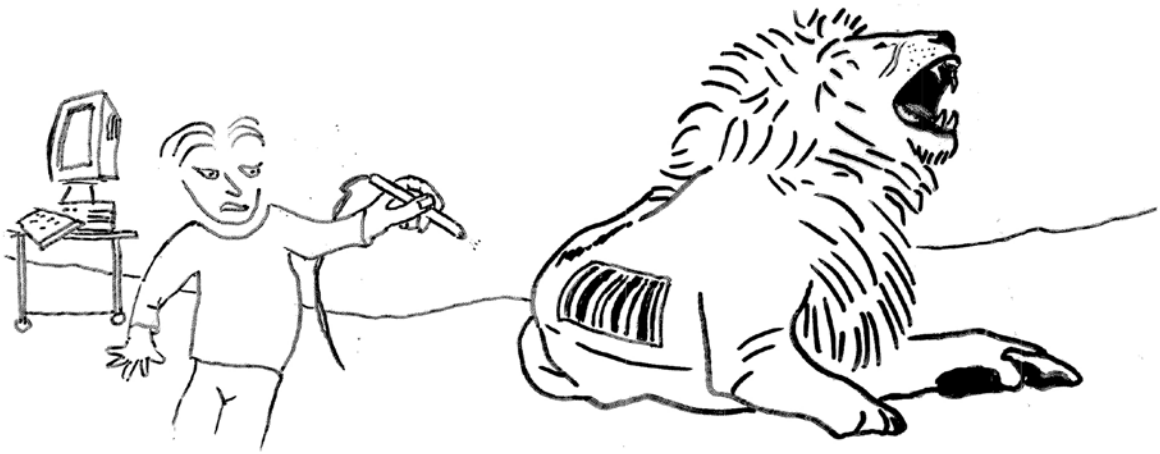
Softwareschule Schweiz

Projekt-Buch : Video-Barcode-Leser



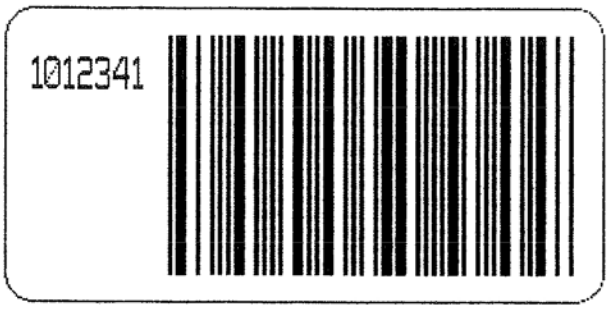


Diplomarbeit T29 / 9. Juni 1994

Autor : Jürg Arnold
Betreuer : Stephan Fischli

Basis-Pflichtenheft

Video-Barcode-Leser



Softwareschule Schweiz

Autor : Jürg Arnold
Betreuer : Stephan Fischli
Klasse : T29
Datum : 9. Juni 1994

Inhalt :

1. Einleitung
   1.1 Sinn und Zweck
   1.2 Umfang und Anwendungsbereich
   1.3 Definitionen und Abkürzungen
2. Allgemeine Beschreibung
   2.1 Uebersicht
   2.2 Beziehungen zu anderen Systemen
   2.3 Uebersicht der Funktionen
   2.4 Allgemeine Einschränkungen
3. Funktionale Anforderungen
   3.1 Allgemeine Funktionen
      3.1.1 Installation
      3.1.2 System Setup
      3.1.3 Normalmodus
      3.1.4 Visualisierungsmodus
   3.2 Funktion TIF-File in Bitmap
   3.3 Funktion Bitmap verarbeiten
   3.4 Fehlermeldungen
      3.4.1 File-Error
      3.4.2 Bild-Error
4. Externe Schnittstellen
   4.1 Softwareschnittstellen
5. Leistungsanforderungen
6. Adaptionen
7. Anhang

1. Einleitung

1.1 Sinn und Zweck

Das Projekt "Video-Barcode-Leser" ist eine Studie, in der abgeklärt werden soll, ob und wie es technisch möglich ist, einen Barcode zu erfassen, der nicht mit einem handels-üblichen Infrarot-Barcodeleser gelesen, sondern von einer Videokamera "fotografiert" und anschliessend mittels Computer ausgewertet wird.

Konventionellen Infrarot-Barcodelesern ist gemeinsam, dass verzerrte oder verschmutzte Barcode-Labels schlecht gelesen werden können. Diese Beobachtung wurde auch bei der Firma TECAN AG gemacht. Dort werden Reagenzflaschen vom Durchmesser 13mm längswegs mit Barcodelabels der Grösse 28mm x 17mm beklebt. Diese Reagenzflaschen werden auf ein Karussel aufgesteckt, das sich um eine zur Grundfläche schiefe Rotationsachse dreht. Das dem IR-Barcode-Leser somit "optisch-verzerrt" präsentierte Barcodelabel konnte nicht immer erfolgreich gelesen werden.

Vorteile der Barcodelesung mit Videokamera/Computer :

- Berührungsloses Lesen
- Barcodepräsentierung lageunabhängig (vorwärts/rückwärts/seitwärts)
- Unterlage des Barcodes beliebig (Barcode auf Orange/Elefant)
- Verschmutzter oder beschädigter Barcode ebenfalls lesbar
- Lesen auf Distanz, zB. Codes auf Eisenbahnwagen (Optik zoombar)
- Suchen des Barcodes auf einer grösseren Fläche und anschl. lesen
- Keine Rotationsteile wie bei den berührungslosen IR-BC-Leser (Wartung !)

Nachteile der Barcodelesung Videokamera/Computer :

- Teurer als IR-Barcodeleser
- Elektronik zur Videobildauswertung (Computer)
- Komplizierte und rechenintensive Algorithmik bei der Bildverarbeitung
- Ausleuchtung des Objekts erforderlich
- braucht mehr elektrische Energie

1.2 Umfang und Anwendungsbereich

Da es in der gesetzten Zeit nicht möglich ist, alle Anwendungsberreiche, die in Kap. 1.1 unter "Vorteile" genannt wurden, zu untersuchen bzw. zu realisieren (Barcode suchen und heranzoomen, usw.), wird auf folgende Anwendung eingeschränkt :

- Berücksichtigung nur des Codabar-Codes (Firma TECAN AG)
- Lesen eines Barcodes von bestimmter Grösse (Barcode zu Gesamtbild)
- Ausreichende Beleuchtung des Barcode-Labels vorausgesetzt
- Barcode darf nicht auf- oder durchgeschnitten sein
- Barcode muss mindestens von Auge lesbar sein
- Klare Abgrenzung Hintergrund zu Barcode
- Ausreichender Kontrast

Folgende Problemfälle sollen abgedeckt werden :

- Lesen eines verschmutzen Barcodes
- Lageunabhängige Lesung
- Teilweise Abdeckung des Barcodes durch Reagenzhalterung

Gesamt-Aufbau :

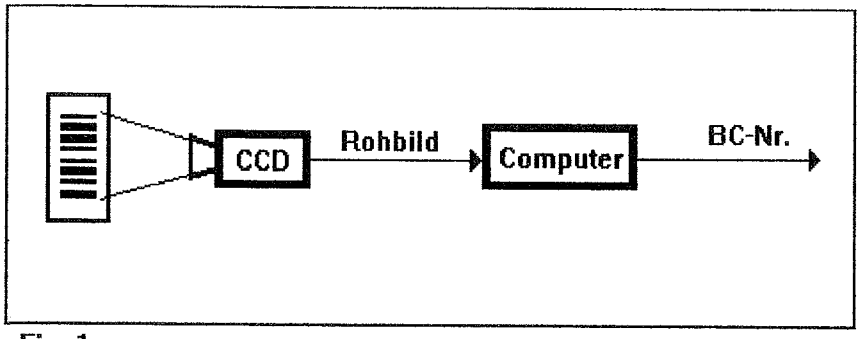


Fig. 1

Der in Fig. 1 gezeigte Gesamt-Aufbau wird auf den interessierenden und wesentlichen Teil-Aufbau (Fig. 2) begrenzt :

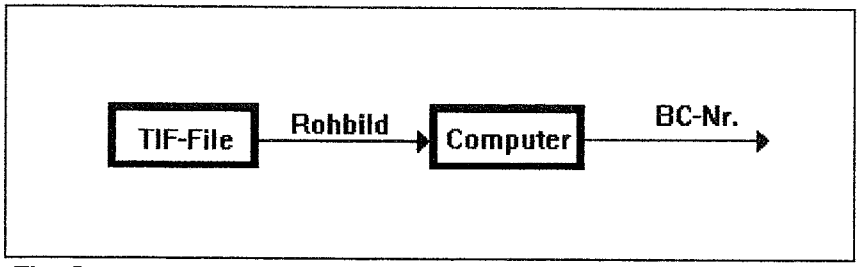


Fig. 2

Ziel ist es, "Videofotografien" von Barcodelabels im TIF-Fileformat zu analysieren und die eindeutige Barcodenummer einem übergeordneten System zur Verfügung zu stellen. Eine Barcodelesung soll die bestimmte Zeitdauer von 3 Sekunden nicht überschreiten.

1.3 Definitionen und Abkürzungen

BC : Barcode
IR : Infrarot
CCD : Charge Coupled Device
TIF : Tag Image File

2. Allgemeine Beschreibung

2.1 Uebersicht

Für die Uebersicht wird bewusst die Darstellungsform "Datenflussdiagramm" gewählt. Obwohl diese Methode normalerweise erst in der Analysephase Anwendung findet, kann damit die Systemgrenze aufgezeigt werden :

Eine zusätzliche Aufgabe, die mit dem eigentlichen Problem der Bildverarbeitung nichts zu tun hat, stellt sich mit der Interpretation des TIF-Fileformates. Diese Funktion wird im Diagramm (Fig. 4) besonders hervorgehoben. Der Prozess "TIF-File in Bitmap"
würde bei der Integration in ein komplettes System mit Videokamera usw. entfallen.

Eigentlicher Kern der Aufgabe ist aber der Prozess "Bitmap verarbeiten". Soll in einem weiteren Projekt die Komplettlösung mit CCD-Kamera realisiert werden, muss das Modul, das die Bildverarbeitung macht, ohne Aenderungen übernommen werden können (vgl. Fig. 4)

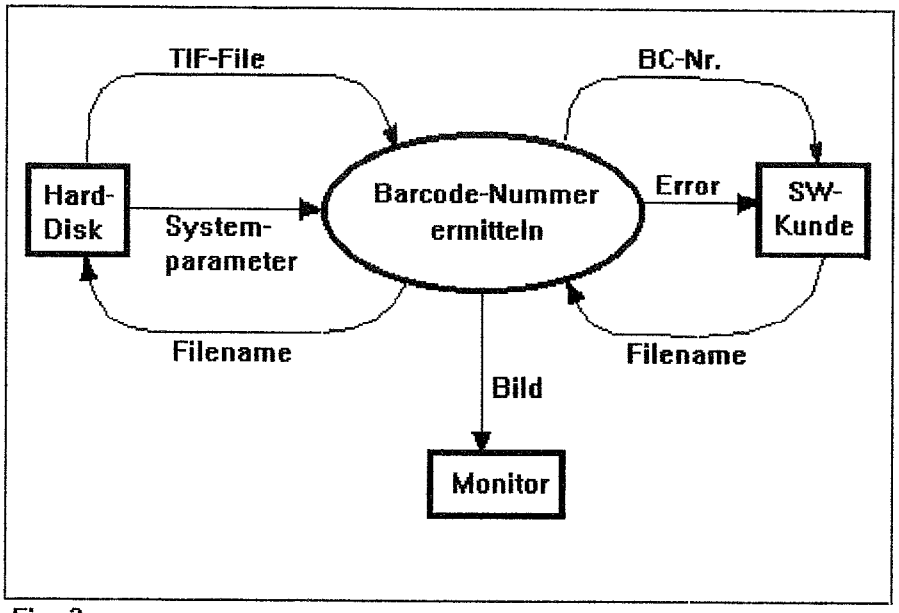


Fig. 3

2.2 Beziehungen zu anderen Systemen

Da der Videoteil nicht Gegenstand dieser Untersuchungen ist, entfällt das Videokamera-System bzw. Frame-Grabber usw.

Der SW-Kunde in Fig. 3 ist eine beliebige Applikation, die auf der selben Hardware wie die Barcodeermittlung läuft und explizit eine Barcodeanfrage (Filename) macht.

2.3 Uebersicht der Funktionen

Aufteilung des Kontextes (Fig. 3) in interne Prozesse : (Ist als Anforderung zu verstehen!!!)

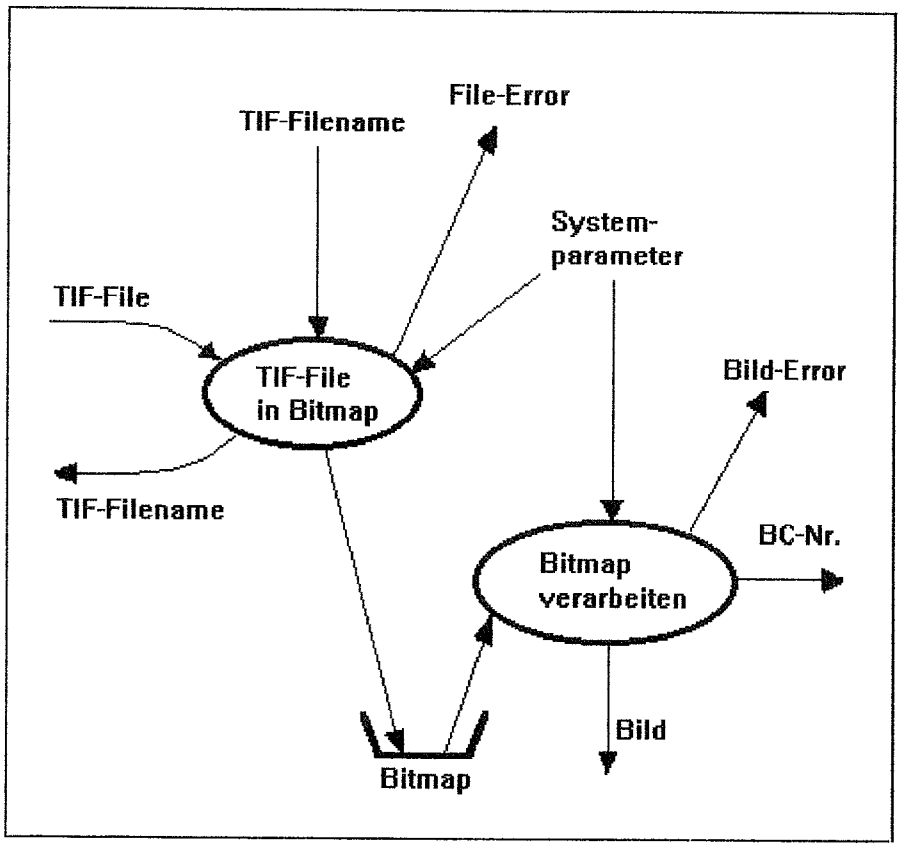


Fig. 4

Funktion "TIF-File in Bitmap"

- Einlesen des vom "SW-Kunden" verlangten TIF-Files ab Harddisk.
- Extrahieren der Tag's im TIF-File
- Anlegen eines Bitmaps im Arbeitsspeicher

Funktion "Bitmap verarbeiten"

- Erkennen des Barcodelabel im eingelesenen Bitmap.
- Interpretieren des Barcodes
- Ausgabe der Barcode-Nummer.

2.4 Allgemeine Einschränkungen

Es werden ausschliesslich Codabar-Codes ausgewertet und zwar nur solche mit der Start-Stop-Zeichenkombination a/t (vgl. Anhang "Der Codabar Code")

Der Barcode-Leseprozess muss fehlerfrei funktionieren. Das heisst, es dürfen keine Falschinterpretationen vorkommen. Sollte eine Lesung Schwierigkeiten bereiten, so ist das Bildmaterial zurückzuweisen (Error). Die Weitergabe einer falschen Nummer hätte in einer Blutbank als Anwendungsgebiet des Codabar-Codes verheerende Folgen (Sicherheitsaspekte).

Es werden ausschliesslich s/w-Bilder ausgewertet, wobei Format und Anzahl Graustufenwerte parametrisierbar sind.

Die Bilder werden in keinem Zeitpunkt komprimiert. Auch das TIF-File soll unkomprimiert vorliegen.

Kritischer Punkt in der Applikation ist die Verarbeitungsgeschwindigkeit, dh.die Wahl von geeigneten Algorithmen und deren zeitliche Optimierung und der enorme Platzbedarf im Arbeitsspeicher für die Zwischenablage der Bitmaps.

3. Funktionale Anforderungen

3.1 Allgemeine Funktionen

3.1.1 Installation

Das Barcodelesemodul wird zur Hauptapplikation (SW-Kunde, siehe 2.1) dazugelinkt. (C-Headerfile !) Eine eigentliche Installation entfällt.

3.1.2 System Setup

Systemparameter zum Einschalten eines Visualisierungstools

Das Lesemodul soll in zwei Betriebszuständen laufen können :

1. Normalmodus (siehe 3.1.3)
2. Visualisierungsmodus (siehe 3.1.4)

Systemparameter für die Bildcharakteristik

Das Bildformat, dh. die Anzahl Pixel in Länge und Breite soll parametrisierbar sein, wie auch die Anzahl der Grauwerte pro Pixel. Defaultwerte sind :

- Bildformat 512 x 512 Pixel
- Anz. Grauwerte 8 Bit/Pixel

Da das TIF-File die Angaben über Bildformat und Graustufen enthält, kann es zu Konflikten zwischen TIF-File-Parameter und Systemparameter kommen. In diesem Fall ist das TIF-File als ungültig zurückzuweisen. Dies hinsichtlich der Komplettlösung mit CCD-Kamera, wo nicht ein TIF-File Ausgangslage zur Bildauswertung ist.

Die Systemparameter sind im Headerfile der Funktion "Bitmap verarbeiten" zu finden. Auf das Einlesen der Systemparameter ab Systemparameter-File wird verzichtet. (Nicht substantiell, könnte nachträglich ohne grossen Aufwand eingebaut werden.)

3.1.3 Normalmodus

Im Normalmodus wird im Gegensatz zum Visualisierungsmodus nur die reine Verarbeitung durchgeführt. Ein Barcode-Bitmap als Eingabe hat eine Barcode-Nummer oder Fehlermeldung als Ausgabe zur Folge.

3.1.4 Visualisierungsmodus

Um die Verarbeitungsschritte (Etappen) der Bildverarbeitung dem interessierten Publikum vorzuführen, soll mit einem Systemparameter Normalmodus oder Visualisierungsmodus eingeschaltet werden können. Im Visualisierungsmodus soll nach jeder Bildveränderung dieses auf einem Bildschirm (Monitor) angezeigt werden. (vgl. Fig. 3)

3.2 Funktion TIF-File in Bitmap

Kurzbeschreibung

- Einlesen des vom SW-Kunden verlangten TIF-Files ab Harddisk
- Extrahieren der TIF-File Tags (unkomprimierte Bilddaten !)
- Anlegen eines Bitmaps im Arbeitsspeicher

Eingaben

- Filename des TIF-Files

Ausgabe

- File-Error im Fehlerfall

3.3 Funktion Bitmap verarbeiten

Kurzbeschreibung

- Bearbeiten des Bitmaps
  - Bildverbesserung
  - Kantendetektion
  - Vektorisierung
  - Mustererkennung
- Die SW-Schnittstelle ist vorgegeben (vgl. 4.1)

Ausgaben

- Nummer des Barcodes oder Bild-Error

3.4 Fehlermeldungen

3.4.1 File-Error

- Fehleingabe Filename ungültig
- TIF-File nicht gefunden
- kein Speicherplatz mehr vorhanden

3.4.2 Bild-Error

- nicht auswertbares Bildmaterial
- kein Speicherplatz mehr vorhanden

Die Fehlermeldungen beim Bild-Error sollen in einer Nummer verschlüsselt werden. (vgl. 4.1)

4. Externe Schnittstellen

4.1 Softwareschnittstellen

Die Schnittstelle der Funktion "Bitmap verarbeiten" wird folgendermassen spezifiziert :

int bc_process(long * BC_Nr, void * Bitmap)

Return-Value :	0 = Fehlerfrei 1 = Bild nicht auswertbar 2 = Kein Speicherplatz mehr vorhanden
long * BC_Nr	Zeiger auf Nummer des Barcodes
void * Bitmap	Zeiger auf ein Bitmap zB. char * Bitmap[512][512] char = 8 Bit

5. Leistungsanforderungen

Eine Bildlesung und Interpretierung innerhalb drei Sekunden !
100 % Sicherheit der Lesung oder Rückweisung des Auftrages mit einer Fehlermeldung

6. Adaptionen

Der Prozess "Bitmap verarbeiten" soll in einem Folgeprojekt wiederverwendet werden können. (SW-Schnittstelle, vgl 4.1 !)

Es bestehen keine besonderen Anforderungen bezüglich Portierbarkeit. Das Lesen des TIF-Files soll IBM-spezifisch gelöst werden. Die Algorithmik (Mathematik) als essentieller Teil der Arbeit ist automatisch portierbar !

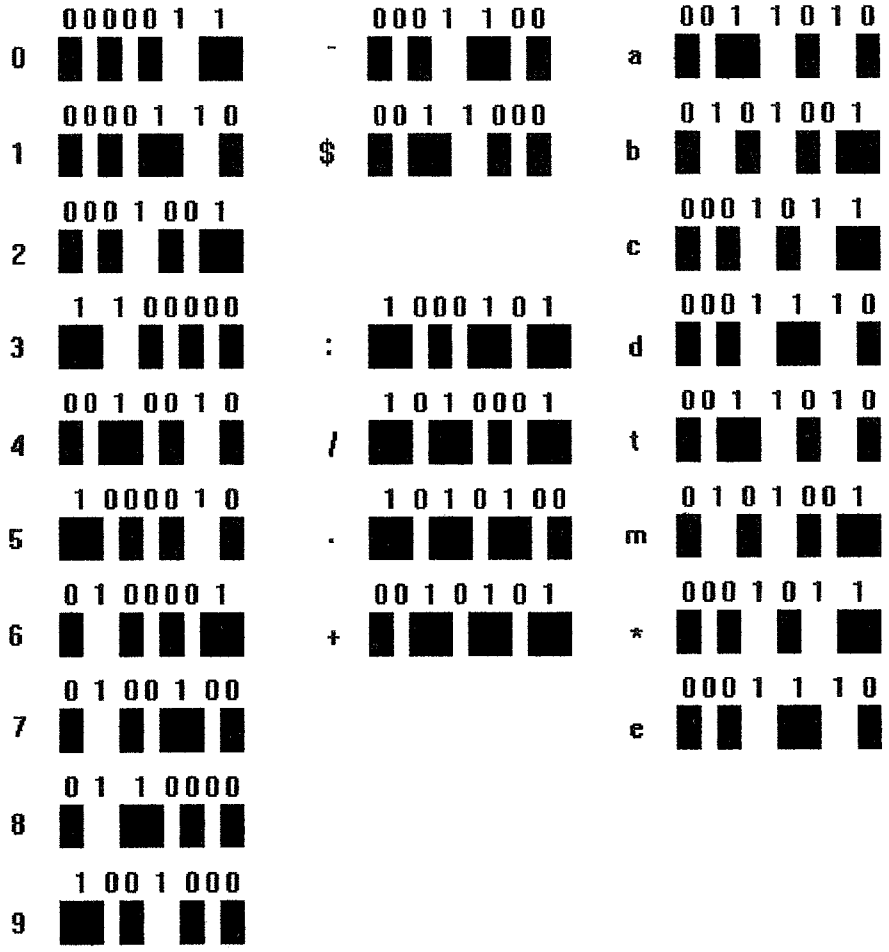
7. Anhang

Der Codabar Code

Der Codabar Code ist ein diskreter Code und stellt die numerischen Ziffern (0..9) und einige Sonderzeichen ($, -. :, /, ., +) zur Verfügung. Der Code wurde speziell für die Anforderungen im Medizinbereich, Krankenhauswesen, Gesundheitswesen und Blutbanken entwickelt. Speziell in diesen Bereichen ist eine hohe Sicherheit erforderlich. Dieser Code verwendet 4 verschiedene Start-Stop-Zeichenkombinationen:

(**a/t , b/m , c/* , d/e**).

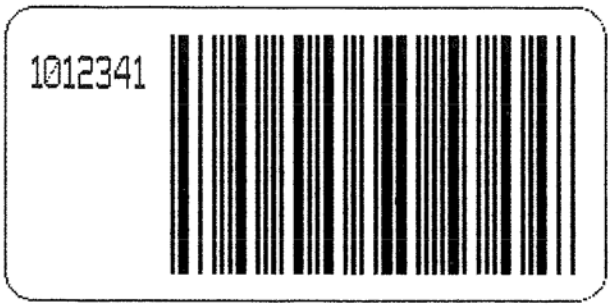
Die Start-Stop-Zeichekombination werden als Schlüssel zu verschiedenen Datenbanken eingesetzt. Wie das folgende Bild verdeutlicht, verwendet der Code 7 Bits (maximal 7 schwarze und weisse Elemente) für die Codierung eines Zeichens.



Die numerischen Ziffern enthalten alle 2 weite Elemente (1 x schwarz, 1 x weiss), die Sonderzeichen 3 weite Elemente (3 x schwarz), die Start-Stop-Zeichen jeweils 2 weisse und ein schwarzes weites Element. Die weiten und schmalen Elemente haben nicht immer die gleiche Breite, es gibt 18 verschiedene Strichbreiten. Die einzelnen Zeichen als Strichcode codiert ergeben dadurch immer die gleiche Breite. Der Codabar Code wird meist mit einer Auslösung von 0.17 mm ausgedruckt. Damit können 11 Zeichen pro Zoll dargestellt werden.

Terminplan

Video-Barcode-Leser



Softwareschule Schweiz

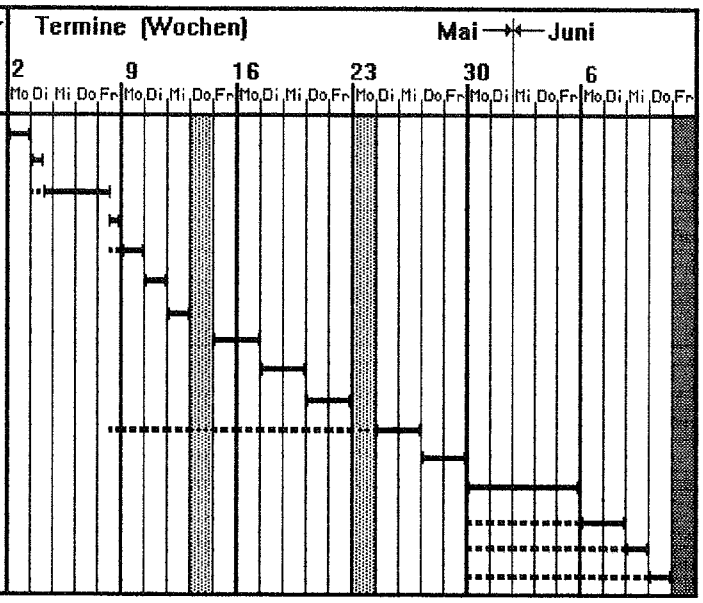
Autor : Jürg Arnold
Betreuer : Stephan Fischli
Klasse : T29
Datum : 9. Juni 1994

1. Arbeitspakete

1. Anforderungsspezifikation
2. Terminplanung
3. Analyse Gesamtprojekt
4. Testkonzept
5. Design TIF-File-Extraktion
6. Design Bildverarbeitung Schritt A "Binärisierung"
7. Design Bildverarbeitung Schritt B "Expandierung"
8. Design Bildverarbeitung Schritt C "Kantendetektion"
9. Design Bildverarbeitung Schritt D "Vektorisierung"
10. Design Bildverarbeitung Schritt E "Mustererkennung"
11. Design MMI für Test und Präsentation
12. Implementation/Integration
13. Systemtest
14. Bereinigte Dokumentation
15. Abschlussbericht
16. Präsentation vorbereiten

2. Balkendiagramm

	Tätigkeit	Vorg.	Dauer
1.	Anforderungen	-	1
2.	Terminplanung	1	0.5
3.	Analyse	1	3
4.	Testkonzept	3	0.5
5.	Design TIF-File	3	1
6.	Design Binaer.	5	1
7.	Design Expand	6	1
8.	Design Kanten	7	2
9.	Design Vektor	8	2
10.	Design Muster	9	2
11.	Design MMI	3	2
12.	Impl/Integr.	10,11	2
13.	Systemtest	4,12	5
14.	Doku bereinigen	13	2
15.	Abschlussbericht	13	1
16.	Praes. vorber.	13	1



3. Diskussion

Die Arbeit findet statt von :

Montag, 2. Mai 1994, 8.00 Uhr

bis :

Donnerstag, 9. Juni 1994, 16.30 Uhr

Zu beachten ist, dass das Auffahrts- Pfingstwochenende in diesen Zeitraum fällt, was eine Reduktion von 2 Arbeitstagen ergibt !

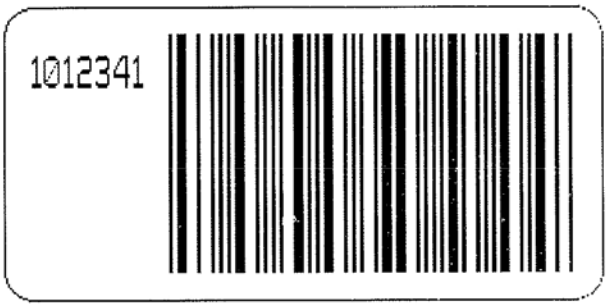
Die Arbeit teilt sich grob in folgende Teile auf :

Analyse	: 4 Tage	=	15%
Design	: 11 Tage	=	40%
Implementation	: 2 Tage	=	7%
Test	: 5,5 Tage	=	20%
Diverses	: 4,5 Tage	=	18%
Total	: 27 Tage	=	100%

Dies entspricht nicht der klassischen Verteilung ! Es ist aber zu beachten, dass der Haupteil die Evaluation/Entwicklung von geeigneten Algorithmen für die Bildverarbeitung sein wird. Mit dem Design der jeweiligen Bildverarbeitungs-Teilschritten ist die Implementation bereits zu einem grossen Teil erledigt, so dass in der Implementation nur noch das Zusammenfügen der einzelnen SW-Teile übrigbleiben sollte.

Zusammenfassung der Abschlussarbeit

Video-Barcode-Leser



Softwareschule Schweiz

Autor : Jürg Arnold
Betreuer : Stephan Fischli
Klasse : T29
Datum : 9. Juni 1994

1. Einleitung

Soll ein verschmutztes, auf zylindrischem Untergrund klebendes, schief in der Landschaft stehendes Barcode-Label von einem herkömmlichen Infrarot-Bar-codeleser interpretiert werden, gelingt dies nicht immer fehlerfrei !

Beobachtet wurde dies in einer Firma für medizinische Laborautomaten :
Dort werden dünne Reagenzfläschchen die längswegs mit Barcode-Labels beklebt sind auf schräg montierte Karussels aufgesteckt. Die dem IR-Barcode-Leser optisch-verzerrt präsentierten Barcode-Labels konnten nicht immer erfolgreich gelesen werden.

Daraus entstand die Idee, die Barcode-Labels mit einer Videokamera aufzunehmen und anschliessend per Computer auszuwerten :

(+) Berührungsloses Lesen
(+) Lageunabhängigkeit Vorwärts, Rückwärts ...
(+) Verschmutzungen herausfilterbar

(-) Spezielle HW erforderlich wie Kamera, Computer => Preis !
(-) Speicher - und rechenintensive Bildverarbeitung => Zeit !

2. Zielsetzung

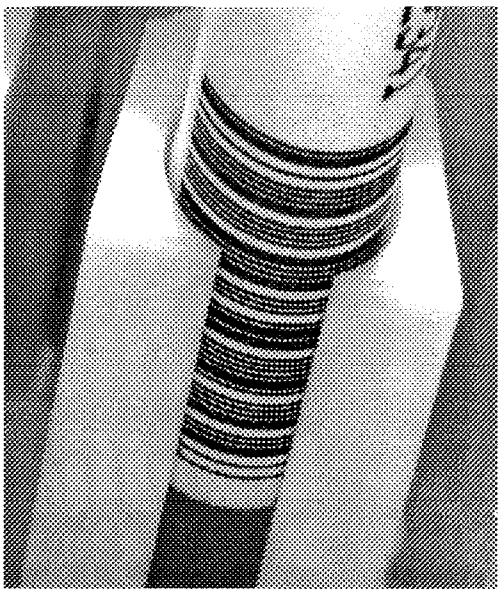


Fig. 1

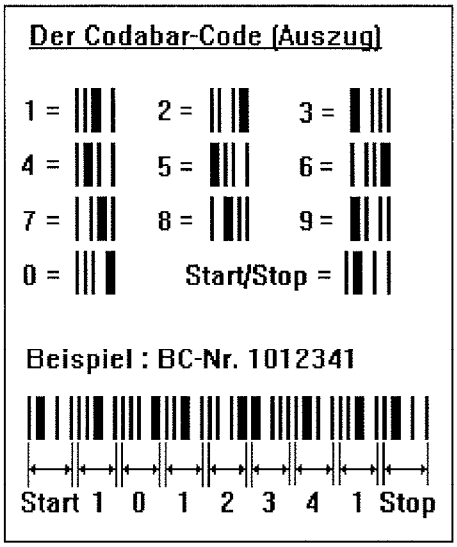
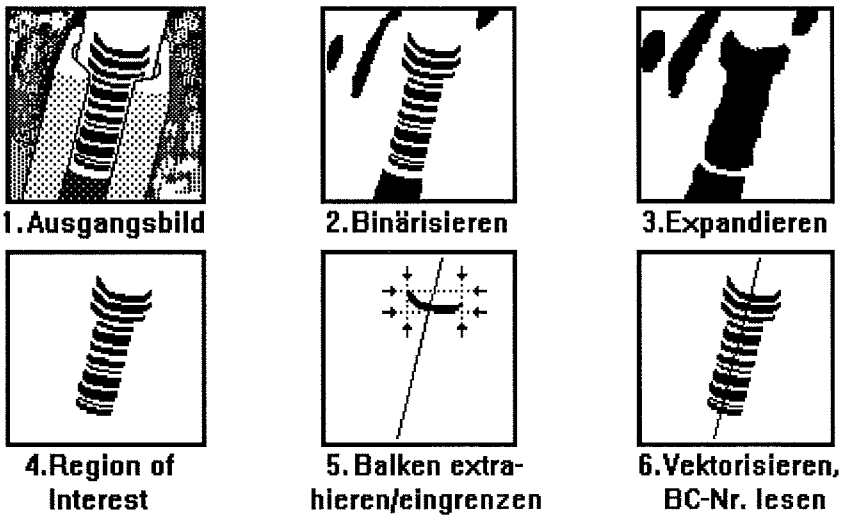


Fig. 2

Da kein Video-Set zur Verfügung steht, soll ein im TIF-Fileformat abgespei-chertes Bild (siehe Fig. 1) gemäss Codabar-Vorschrift (siehe Fig. 2) verarbeitet werden. Eine Auswertung soll nicht länger als 3 Sekunden dauern. In einem "Show-Modus" soll jeder Verarbeitungsschritt auf dem Bildschirm angezeigt werden können.
Vorgängig werden 27 Barcodes in verschiedener Lage und Grösse mit einer Videokamera der Auflösung 512 x 512 Pixel bzw. 256 Grauwerte aufge-nommen und als TIF-File abgespeichert.

3. Realisierung

Das Programm lädt zuerst ein TIF-File ab Harddisk in das Memory.
Danach wird das Bild in 6 Teilschritten verarbeitet :

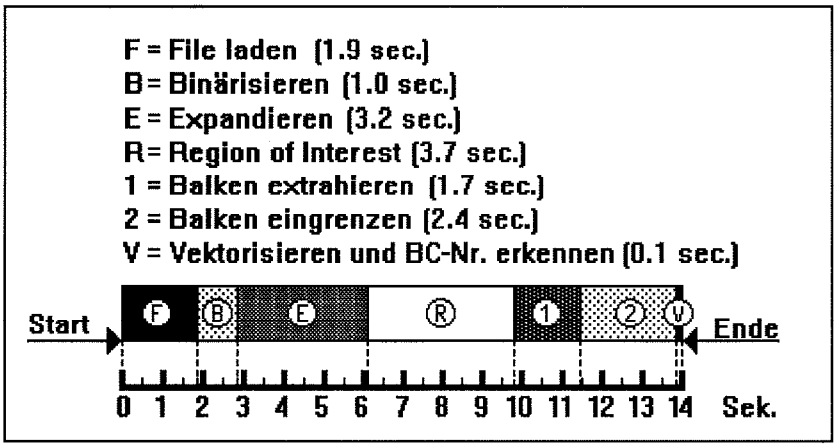


Beim Binärisieren wird im Grauwerteberich des Ausgangsbildes (Histogramm) eine geeignete Schwelle gesucht und das Grauwert-Bild in ein Schwarz/Weiss-Bild umgewandelt. Um den Balkenblock erkennen zu können, wird das Bild expandiert und anschliessend der interessierende Teil herausgeschnitten. Ein speziell ausgesuchter Balken dient dazu, eine geeignete Lesegerade zu finden.

4. Besprechung

Das Programm funktioniert und erkennt 55% der bereitgestellten BC-Bilder.
Bei den nicht erkannten Bilder sind entweder die Codes unleserlich, nicht im Bildzentrum oder beim Expandieren verläuft der Barcode-Flecken in einen angrenzende Umgebungs-Flecken, was zu Fehlermeldungen führt.

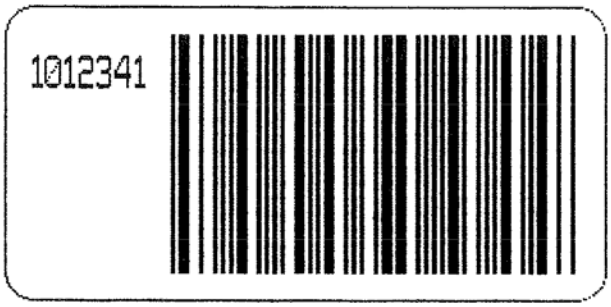
Ein weiters Problem ist die Verarbeitungszeit von 14 Sekunden :



Bei Region-of-Interest könnte mit raffinierterem Algoritmus noch ca. 1 Sek. herausgeholt werden. Bessere Ausgangsbilder bräuchten weniger Verarbeitungsschritte. Ev. Bildvorverarbeitung auf spez. HW, Multiprozessing.

Projekt-Bericht

Video-Barcode-Leser



Softwareschule Schweiz

Autor : Jürg Arnold
Betreuer : Stephan Fischli
Klasse : T29
Datum : 9. Juni 1994

Inhalt

1. Einleitung
   - 1.1 Zweck
   - 1.2 Grundlage
   - 1.3 Definitionen und Abkürzungen
2. Konzept
   - 2.1 Hardware-Architektur
   - 2.2 Software-Architektur
3. Systementwurf
4. Programmbeschreibung/Realisierung
   - 4.1 Ausgangsbild lesen
     - 4.1.1 Das TIF-Fileformat
     - 4.2.2 Speicherverwaltung
   - 4.2 Binärisieren
   - 4.3 Expandieren
   - 4.4 "Region of Interest" ermitteln
   - 4.5 Balken extrahieren und eingrenzen
   - 4.6 Vektorisieren und BC-Nr. lesen
5. Test/Besprechung
6. Wie weiter ?
7. Programmbenutzung
8. Literaturverzeichnis
9. Anhang : Headerfiles
10. Anhang A (TIF-Bilder)

1. Einleitung

1.1 Zweck

Soll ein verschmutztes, auf zylindrischem Untergrund klebendes, schief in der Landschaft stehendes Barcode-Label von einem herkömmlichen Infrarot-Barcodeleser interpretiert werden, gelingt dies nicht immer fehlerfrei !

Beobachtet wurde dies in einer Firma für medizinische Laborautomaten :
Dort werden dünne Reagenzfläschchen die längswegs mit Barcode-Labels beklebt sind auf schräg montierte Karussels aufgesteckt. Die dem IR-Barcode-Leser optisch-verzerrt präsentierten Barcode-Labels konnten nicht immer erfolgreich gelesen werden.

Daraus entstand die Idee, die Barcode-Labels mit einer Videokamera aufzunehmen und anschliessend per Computer auszuwerten :

(+) Berührungsloses Lesen
(+) Lageunabhängigkeit Vorwärts, Rückwärts ...
(+) Verschmutzungen herausfilterbar

(-) Spezielle HW erforderlich wie Kamera, Computer => Preis !
(-) Speicher - und rechenintensive Bildverarbeitung => Zeit !

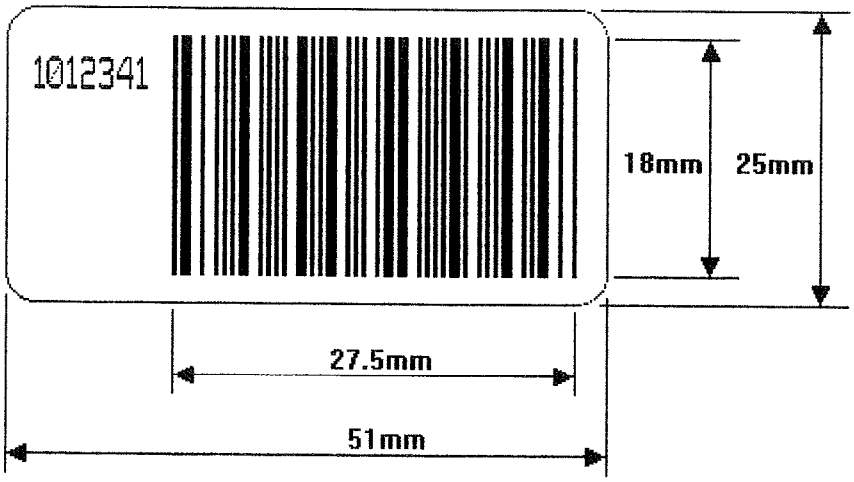
1.2 Grundlage

Da kein Video-Set zur Verfügung steht, soll ein im TIF-Fileformat abgespeichertes Bild (siehe Anhang A, Fig. 1 bis Fig. 27) gemäss Codabar-Vorschrift verarbeitet werden. Eine Auswertung soll nicht länger als 3 Sekunden dauern. In einem "Show-Modus" soll jeder Verarbeitungsschritt auf dem Bildschirm angezeigt werden können.

Die Ausgangsbilder (Anhang A.) wurden vorgängig in verschiedener Lage und Grösse mit einer Videokamera der Auflösung 512 x 512 Pixel bzw. 256 Grauwerte aufgenommen und als TIF-File abgespeichert.

Es werden nur Codabar-Codes ausgewertet.

Ein Beispiel für ein Codabar-Label :



Dies sind selbstklebende Etiketten, die auf Reagenzröhrchen aufgeklebt werden.

1.3 Definitionen und Abkürzungen

TIF	=	Tag Image File
BC	=	Barcode
Pixel	=	Picture element => Bildpunkt
Binär	=	Zwei Zustände (zB. Binärbild ist ein Bild mit nur noch zwei Farben/Graustufen wie Schwarz und Weiss)
CCD	=	Charge-Coupled-Device, lichtempfindlicher Sensor
Grauwert	=	Zahlenwert, der die Helligkeitsintensität eines Bildpunktes repräsentiert
Histogramm	=	Die graphische Darstellung der Häufigkeit der einzelnen Grau- oder Farbwerte
Segmentierung	=	Aufteilung einer Szene in verschiedene Bereiche nach bestimmten Kriterien. Dabei werden implizit diesen Bereichen schon Eigenschaften zugeordnet
Vektor	=	Eine Strecke mit einer Richtung und mindestens 2 Pixel länge

2. Konzept

Da im voraus noch ungewiss war, welche Verarbeitungsschritte und Algoritmen Anwendung fänden, konnte keine SW-Analyse bzw. SW-Design im klassischen Sinne durchgeführt werden. Ein Ueberblick über das HW und SW-Konzept sollen nun folgende Abschnitte geben.

2.1 Hardware-Architektur

Die Applikation soll auf IBM-DOS-Rechner ab 80486DX mit mind. 33MHz Systemclock laufen.
Im Showmodus wird nur VGA unterstützt (VGA = 480 x 640)

2.2 Software-Architektur

Rechnerbetriebssystem	:	MS-DOS ab V6.0
Programmiersprache	:	Borland C++, V3.1
Portierbarkeit	:	Die Programmiersprache C rühmt sich, eine leicht portierbare Programmiersprache zu sein. Anforderungen an die Portierbarkeit werden in diesem Programm aber keine ausdrücklich gestellt. Der gem. Basis-Pflichtenheft geforderte "Show-Modus" dh. Modus, bei dem die Bildverarbeitungszwischenschritte grafisch anzuzeigen sind, macht die Applikation sowieso HW-abhängig (Graphikroutinen) Da aber diese Graphikroutinen nur "Nebensache sind, ist für diesen Teil Portierbarkeit sicher kein Thema. Die Verarbeitungsroutinen sind aber in weiten Teilen portierbar ausser der Memory-Allozierung, die wegen der speziellen Speichersegmentierung von MS-DOS etwas exotisch ausfällt.

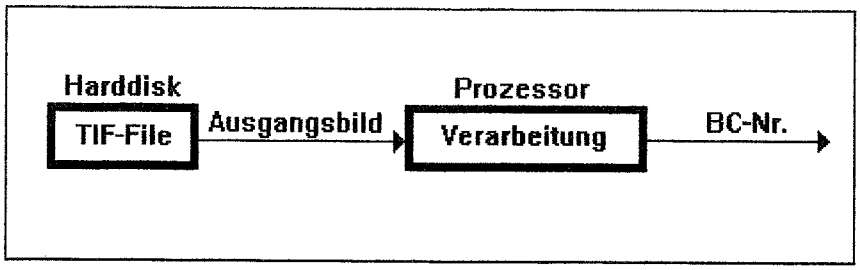


Fig. 1

Der Verarbeitungsprozess (siehe Fig. 1) wird so ausgelegt, dass er später in eine Umgebung mit Videohardware integriert werden kann. Dass heisst, relevanten Verarbeitungsschritte sind diejenige ab dem Zeitpunkt, wo das BC-Bild als Bitmap im Memory abgelegt ist. Die Routinen für das TIF-File lesen und interpretieren sind somit nur "notwendiges Uebel" und nicht Gegenstand von besonderem Interesse.

Die SW-Schnittstelle soll so aussehen :

```
int BC_Process(char far ** fs,     /* Bitmap    */
               char BC_Nr[]);      /* BC-Nummer */
```

bzw.

```
int BC_ProcessShow(char far ** fs, /* Bitmap    */
                   int xb_size,    /* x-Fenster */
                   int yb_size,    /* y-Fenster */
                   char BC_Nr[]);  /* BC-Nummer */
```

für die Version im "Show-Modus".

In einem Systemparameter-Headerfile werden wichtige Konstanten abgelegt, die für die ganze Applikation Gültigkeit haben und auch überall verfügbar sein müssen. Unter anderem :

```
ANZ_PIXEL_IN_X /* Anz. Pixel in x-Richtung */
ANZ_PIXEL_IN_Y /* Anz. Pixel in y-Richtung */
```

Das TIF-File im Harddisk muss ein bestimmtes Format ausweisen, sonst wird es vom TIF-File-Lader nicht akzeptiert :

- 256 Graustufen
- Nicht komprimiert
- 512 x 512 Pixel Auflösung

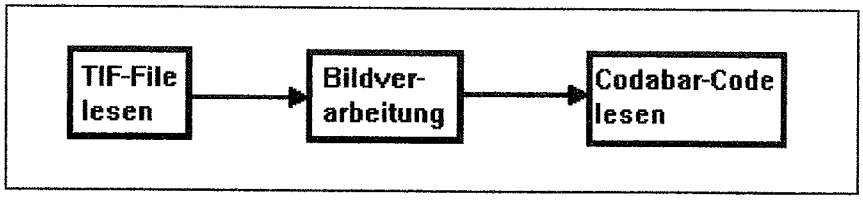


Fig.2

Drei Haupteile soll das Programm vorweisen (Fig. 2) :

1. TIF-File lesen : Interpretieren der Tag's und "abspitzen" des Bitmaps in das Memory. Sollte das TIF-File bzw. deren Tag's nicht dem geforderten Format entsprechen, so wird die Verarbeitung mit einer Fehlermeldung abgebrochen.
2. Das Bild so aufarbeiten, das im Bild nur noch die Balkenzeichnung vorhanden ist und keine Schatten, Umrandungen usw. mehr auftreten.
3. Balkenzeichnung gem. Codabar-Code-Vorschrift in die Nummer konvertieren.

3. Systementwurf

Aufteilung in Module :

SYSPARAM	:	Systemparameter
BC_CST	:	Systemkonstanten
MEMALLOC	:	Speicherverwaltungsroutinen Dynamische Allozierung von Speicherblöcken. Vgl. 4.1.2 Speicherverwaltung
TIF_PARS	:	TIF-File-Zerlegungsroutinen Lesen und interpretieren des TIF-Files. Vgl. 4.1.1 Das TIF-File-format
IMG_PROC	:	Bildverarbeitungsroutinen Routinen zum Binärisieren (4.2), Expandieren (4.3), "Region of Interest" ermitteln (4.4), Balken extrahieren und eingrenzen (4.5)
BC_MATH	:	Vektorgeometrieroutinen Routinen für LineareRegression (4.5), das Finden von Mittellinie und Leselinie (4.6)
CODABAR	:	Codabar-Interpretationsroutine Routine zur Ermittlung der Barcode-Nummer(4.6)
BC_PROC	:	Zusammenfassung aller Verarbeitungsschritte im Normal-Modus
BC_GRAPH	:	Graphikroutinen für ShowModus Routinen zur Anzeige der Bitmaps und Geraden im Graphik-Modus (VGA-Modus 480 x 640)
BC_SHOW	:	Zusammenfassung aller Verarbeitungsschritte im Show-Modus

Die beiden Module BC_GRAPH und BC_SHOW werden einfach weggelassen, wenn es gilt den Leser in eine grössere Applikation zu integrieren.

Die Erklärungen zu den einzelnen Funktionen der Module findet man im Anhang "Headerfiles".

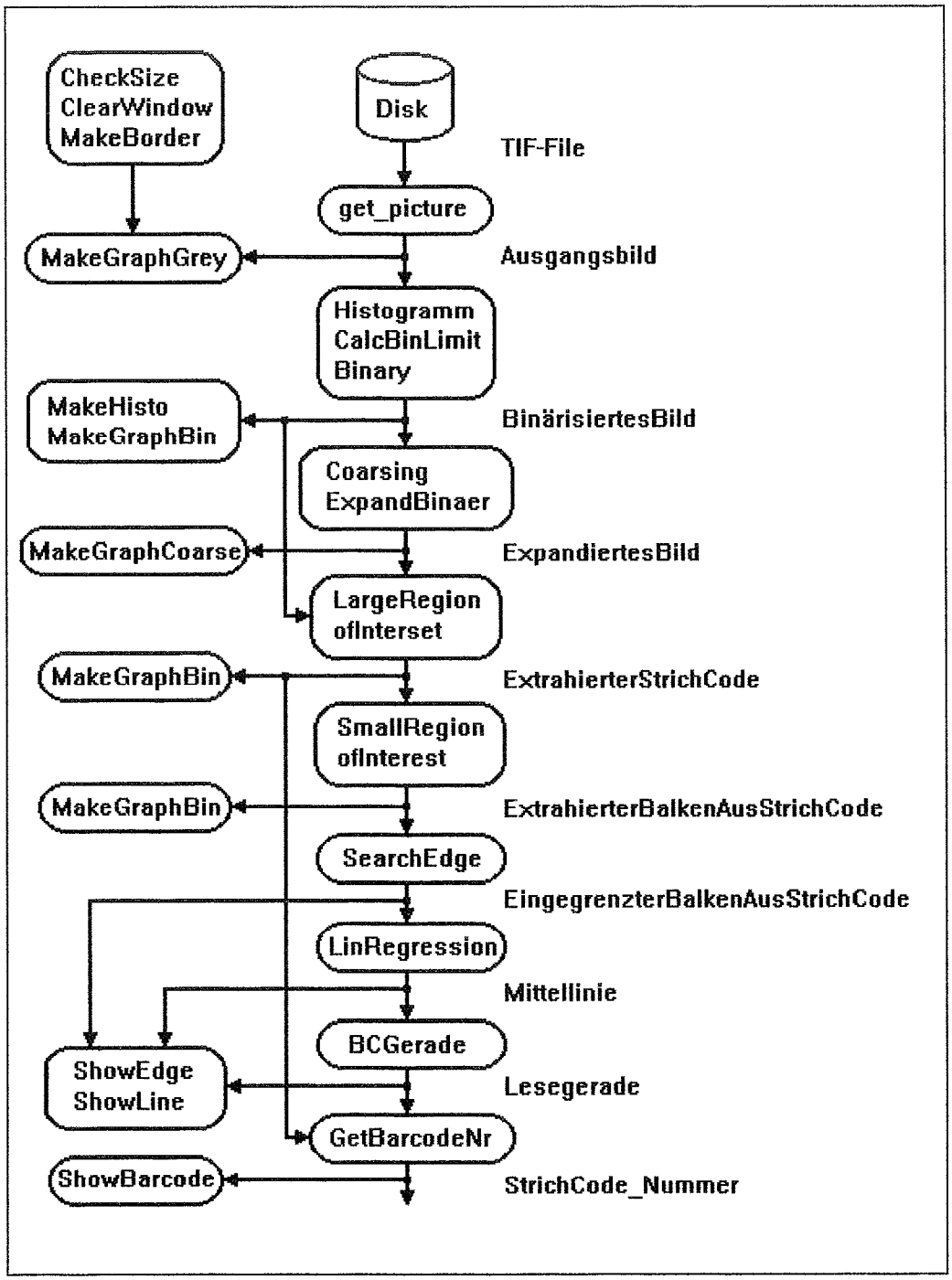


Fig. 3a

Fig. 3a zeigt das Datenflussdiagramm. Die Prozesse der linken Seite wie CheckSize, ClearWindow, MakeGraphGrey, MakeHisto, MakeGraphBin, MakeGraphCoarse, ShowEdge, ShowLine, ShowBarcode werden nur im Show-Prozess-Modus aktiv.

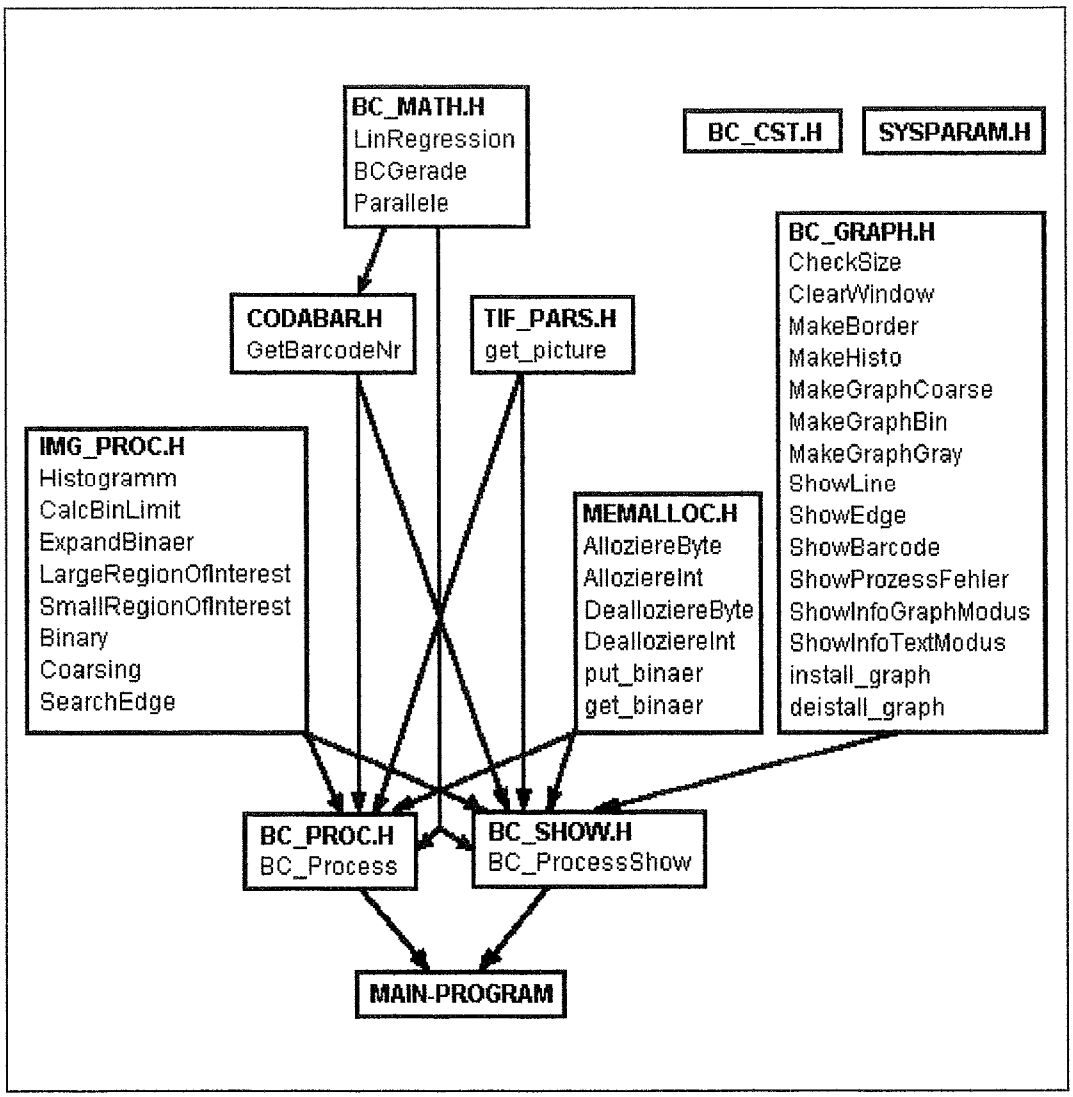


Fig. 3b

Die Modulübersicht in Fig. 3b zeigt, in welchem Modul welche Funktion zu finden ist und welches Modul von welchem Modul im Sinne einer funktionalen Abhängigkeit Verwendung findet.

4. Programmbeschreibung/Realisierung

Uebersicht :

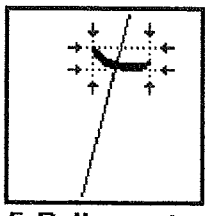
Das Programm lädt zuerst ein TIF-File ab Harddisk in das Memory.
Danach wird das Bild in 6 Teilschritten verarbeitet :

1.Ausgangsbild

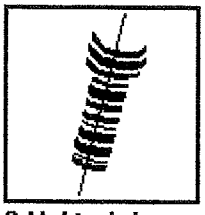
2.Binärisieren

3.Expandieren

4.Region of Interest



5. Balken extrahieren/eingrenzen



6.Vektorisieren, BC-Nr. lesen

Beim Binärisieren wird im Grauwerteberich des Ausgangsbildes (Histogramm) eine geeignete Schwelle gesucht und das Grauwert-Bild in ein Schwarz/Weiss-Bild umgewandelt. Um den Balkenblock erkennen zu können, wird das Bild expandiert und anschliessend der interessierende Teil herausgeschnitten. Ein speziell ausgesuchter Balken dient dazu, eine geeignete Lesegerade zu finden.

Nun die einzelnen Schritte im Detail ...

4.1 Ausgangsbild lesen

4.1.1 Das TIF-Fileformat

Dieses Format ist zurzeit das bedeutenste Bildformat, weil es erlaubt, Bildmaterial zwischen verschiedenenen Computersystemen auszutauschen. Das Dateiformat besteht aus einem Header und den Bilddaten. Die eigentlichen Informationen über den Aufbau des Bildes wie Bildgrösse, Farbtiefe etc. sind nicht im Header, sondern in sogenannten Tags untergebracht.

Beispiel eines Hexdumps einer TIF-Datei :

49 49
2A 00
1E 00 00 00
usw.

Das Format der Datenspeicherung berücksichtigt die prozessorspezifischen Eigenheiten und ist für den Austausch von Daten zwischen Computer mit Intel-Prozessor (zB. IBM) oder Motorola-Prozessor (zB. MacIntosh) essentiell. Intel-Prozessoren legen Daten in der Reihenfolge "Low-Byte - High-Byte" ab, Motorola-Prozessoren umgekehrt. Ein 16-Bit Wort mit dem Wert 1 wird von einem Intel-Prozessor als 01 00, von einem Motorola-Prozessor dasgegen als 00 01 im Rechnerspeicher und somit auch in der Datei abgeglegt. Bei einem 32-Bit-Doppelwort erfolgt die Speicherung bei Intel-Prozessoren konsequenterweise als "Low-Word - High-Word", wobei jedes Wort wieder als "Low-Byte - High-Byte" gespeichert wird. So wird zB. der Wert 65538 (0X10002) als 02 00 01 00 abgespeichert. Nach dem Laden dieser 4 Bytes als Doppelwort in ein 32-Bit-Prozessorregister haben wir dann wieder die Reihenfolge 00 01 00 02 .

Obiger Hexdump nach Sortierung inkl. Bedeutung :

Header :

49 49	: Intel-Format (somit "Low-Byte - High-Byte")
00 2A	: Versionsnummer
00 00 00 1E	: Offset auf den Datenbereich

Datenbereich mit Tags (kann sich an beleibiger Position befinden !) :
Die wichtigsten Tags :

Tag 256 : ImageWidth (Bildbreite)
Tag 257 : ImageLength (Bildhöhe)
Tag 256 : BitsPerSample (Farbtiefe)
Tag 259 : Compression (Art der Datenkompression) (gem. Basispflichtenheft darf das File nicht komprimiert sein !)

Die C-Funktion die das TIF-File verarbeitet, heisst "get_picture", zu finden in TIF_PARS.H.

Für weitergehende Erklärungen zum TIF-Fileformat siehe Lit. [A] !

4.1.2 Speicherverwaltung

Es werden Bilder der Grösse 512 Pixel x 512 Pixel verarbeitet. Jedes Pixel besitzt ein 8-bitiger Grauwert dh. 256 verschiedene Grauwerte pro Pixel.
Speicherbedarf pro Bild : 512 x 512 = 262'144 Byte oder 2'097'152 Bit
Die etwas spezielle Speicheradressierung unter DOS muss daher berücksichtigt werden !

Speicherverwaltung uner DOS :

Total Adressleitungen	: 20
Physikalische Adresse	: 20 Bit
Total adressierbarer Speicherbereich	: 1MByte (1'048'576 Bit)
Vom Programm max. nutzbarer Speicherbereich	: 640kByte

Die Speicheradresse wird in eine Offsetadresse und Segmentadresse aufgeteilt !

Offsetadresse	: 16 Bit
Segmentadresse	: 16 Bit
Speicherbereich bei 16 Bit	: 65'536 Bit

Die Offset- und Segmentadressen überschneiden sich um 12 Bit ! Vgl. Lit. [F] !

Bei Hochsprachen unter DOS wird zwischen Near-Pointern und FAR-Pointern unterschieden. Near-Pointer geben lediglich die Offsetadresse eines Objektes an und sind deshalb auch nur 16 Bit breit. Far-Pointer hingegen bestehen aus einer Segment- und Offsetadresse und werden deshalb als 2 Words abgespeichert.

Der Speicheroperator calloc kann eine maximale Speicherblockgrösse von 64 kByte allozieren. Grössere Speicherblöcke müssen mit farcalloc alloziert werden. Die Funktionen "AlloziereByte", AlloziereInt" bzw. DeallozieреByte" und "DeallozieреInt" allozieren/deallozieren eine Byte- resp. eine Integer-Matrix mit den Ausmassen "zeilen x spalten"

Ein weiteres pikantes Detail stellt der kleinste Datentyp dar :
Dieser ist das char (Charakter) und entspricht 8 Bit oder 1 Byte.
Das ab TIF-File eingelesene Barcodebild wird in eine Charaktermatrix der Grösse 512 x 512 geladen. Da jedes Pixel eine 8-bitige Graustufe hat, ist der Datentyp char somit der geeignete. In einem weiteren Verarbeitungsschritt wird das Graubild in ein Schwarz/Weiss-Bild umgewandelt. Da bei solch grossen Bildern sparsam mit dem Speicher umgegangen werden soll, muss beachtet werden, dass pro Pixel nun nicht mehr 8 Bit wie beim Grauwertbild, sondern nur noch 1 Bit benötigt werden. Dazu dienen die Funktionen put_binaer um ein Bit in den Speicher zu schreiben, bzw. get_binaer um ein Bit aus dem Speicher zu lesen.

Zu finden sind die C-Routinen in MEMALLOC.H !

4.2 Binärisieren

Bei diesem Verarbeitungsschritt gilt es, aus dem Grauwertbild ein Schwarz/ Weiss-Bild zu machen. Man nennt das Binärisieren. Dies wegen den zwei Zuständen Schwarz oder Weiss die ein Pixel annehmen kann. Untersuchungen an den 27 zur Verfügung stehenden Barocdebildern haben gezeigt, dass die optimale Schwelle Schwarz/Weiss nicht bei allen Bildern am selben Ort liegt. Es stellt sich nun die Frage, bei welchem Graustufenwert die Grenze zwischen Schwarz und Weiss gezogen werden soll. Dazu dient das Histogramm. Dies ist die Darstellung der Häufigkeit der einzelnen Grauwerte. Es sollte zwei Maximas enthalten : das eine eher auf der Weiss-Seite das andere eher auf der Schwarz-Seite.

Nun ist es nicht günstig ein Histogramm über das ganze Bild zu machen, wie das auch wieder Tests an den Ausgansbildern gezeigt haben. Da etliche Bilder in den Randzonen diverse Grauwerte aufweisen (Hintergrund Schatten usw.) gehen die beiden Maximas bei Weiss und Schwarz buchstäblich im Rauschen unter. Darum wurde ein Ausschnitt von 170 x 170 Pixel gewählt, um daraus das Histogramm zu erstellen..

Bei einigen Bildern war der Barcode mit Schmutzspuren überdeckt (siehe Anhang A Fig. 3 und Fig.4) Durch eine günstige Wahl der Binärisierungs-schwelle konnten diese Flecken weggefiltert werden.

Die Verarbeitung in Schritten :

Schritt 1 : Bildauschnitt 170 x 170 Pixel im Zentrum des Bildes
Dies ergibt ca. 11% der gesamten Bildfläche.

Schritt 2 : Erstellen des Histogramms
x-Achse = Grauwert [0..255]
y-Achse = Häufigkeit des entsprechenden Grauwertes
Suchen der beiden Maximas
Von den beiden Maximas wird das rechtere (näher beim Grauwert Weiss) ausgewählt und der linke Fusspunkt als Binärisierungsschwelle ausgewählt.

Schritt 3 : Umrechnen des gesammten Grauwertbildes in ein Schwarz/Weiss-Bild.

Siehe Fig. 4 !

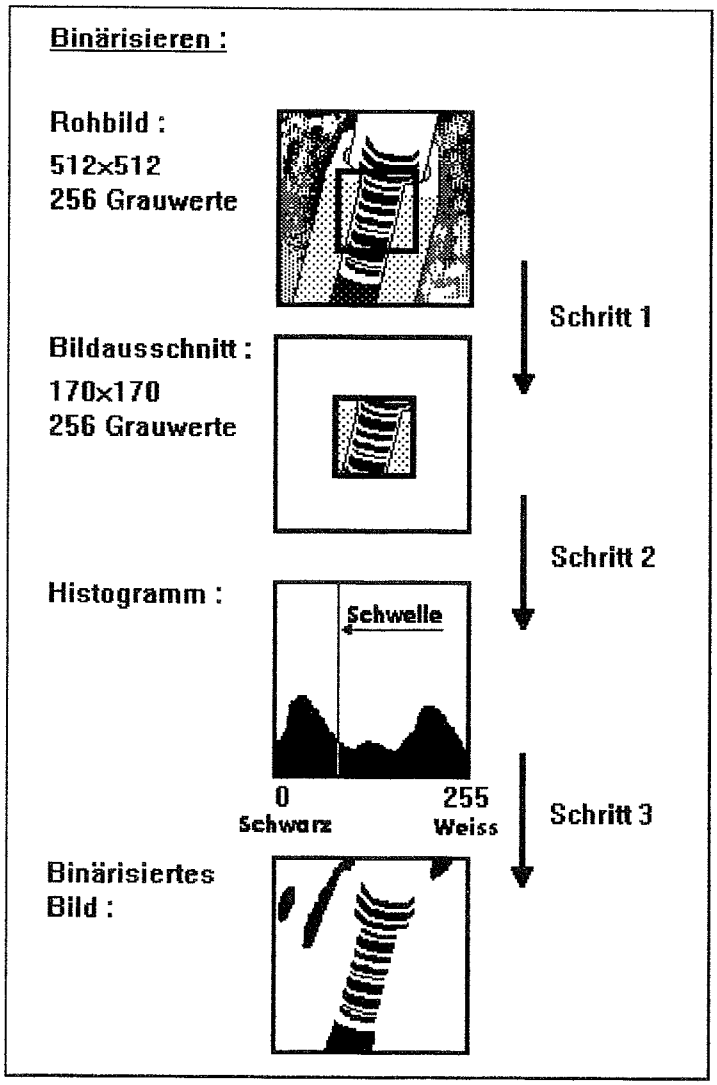


Fig. 4

Die C-Funktion die das Histogramm ermittelt, heisst "Histogramm", die Schwelle berechnet "CalcBinLimit", beide zu finden in IMG_PROC.H !

4.3 Expandieren

Ziel ist es, das Ausgangsbild des Barcodes so aufzubereiten, das alle nicht interessierenden Teile wie Hintergrund, Schatten usw. wegmaskiert sind. Dazu muss der Barcodeteil als Ganzes erkannt werden, um dann alle nicht dazugehörenden Bildteile wegschneiden zu können.

Beim Expandieren werden die Bildteile künstlich aufgeblasen. Die Balken des Strichcodes sollen ineinander verlaufen, so dass sie eine geschlossene Fläche bilden.

Wie aus Fig. 6 ersichtlich ist, braucht jede Schicht die beim Expandieren aufgetragen wird einen Durchlauf durch das ganze Bild. Um dies rationeller geschehen zu lassen, wird das Bild vorerst gröber gerastert. Dass heisst, jeweils 4 Pixel werden zu einem einzigen zusammengefasst (siehe Fig. 5)

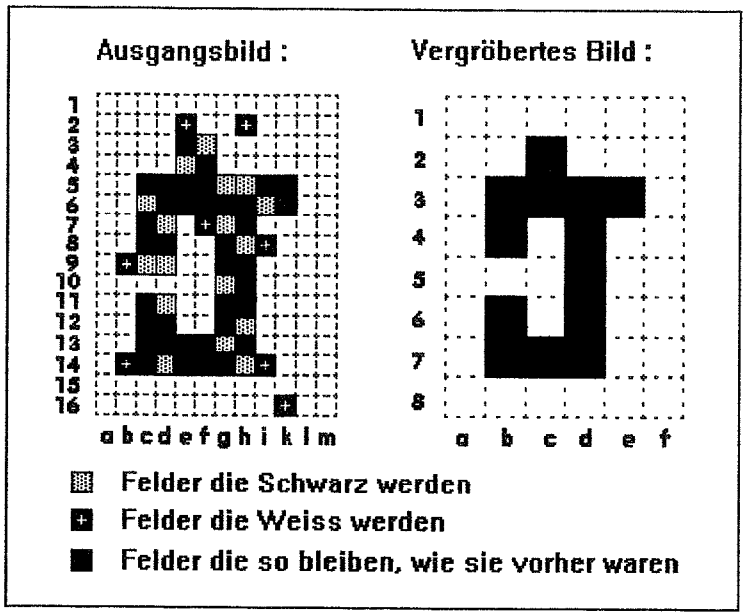


Fig. 5

Nach dem Vergröbern (in der C-Source "Coarsing" genannt) kann an das Expandieren gegangen werden. (in der C-Source "ExpandBinaer")

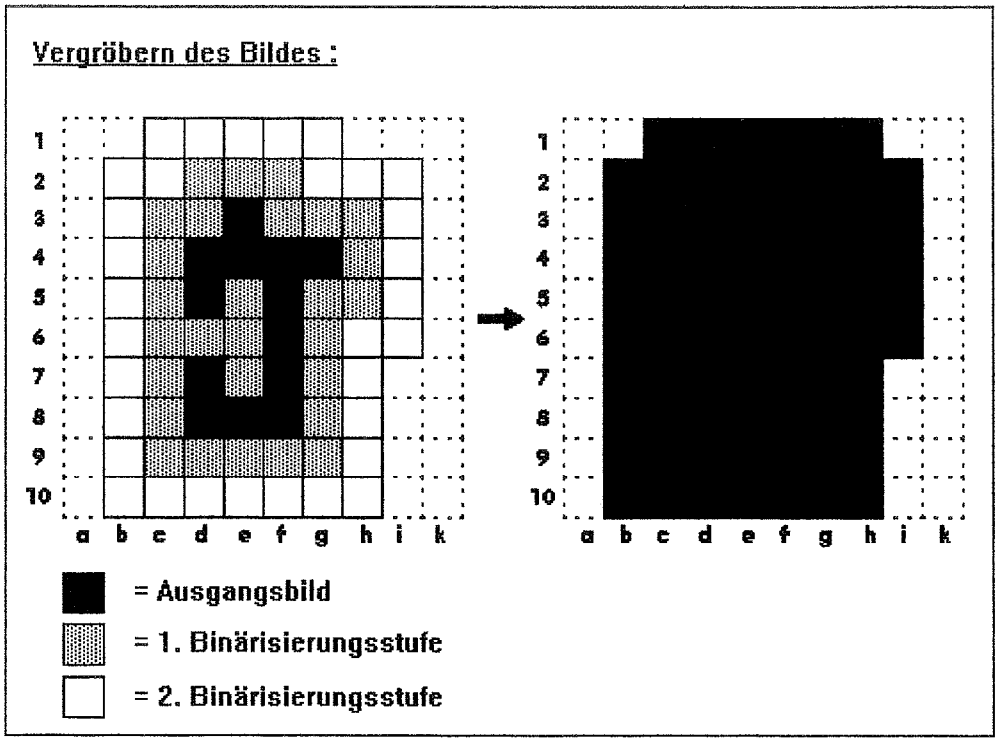


Fig. 6

4.4 "Region of Interest" ermitteln

Der Expandierschritt ergibt ein Bild das ungefähr Fig. 7 entspricht. Als nächstes gilt es, den "Fleck" auszuwählen, unter dem sich der Barcode versteckt. Es wird angenommen, dass es sich um den mittleren handelt !

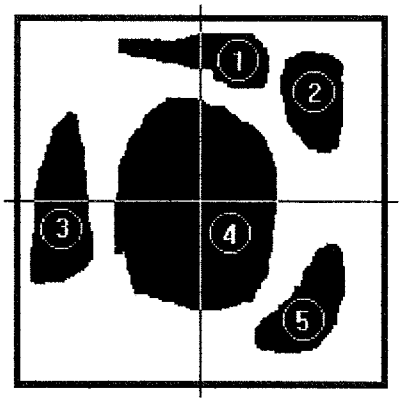


Fig. 7

Naheliegend wäre nun, aus der Mitte rekursiv ein Suchalgoritmus zu starten, der sich auf alle Seiten bis an den Fleckenrand vorarbeiten würde.
Dieser Lösungsansatz scheitert an der zu erwartenden Rekursionstiefe !

Unter der Annahme das der Barcode mit dem Längen zu Breiten-Verhältnis von 1 : 1,5 möglichst gross abgebildet ist, ergibt sich eine bedeckte Fläche von : 512 x 341 Pixel = 174'592 (66%)
Eine solche Rekursionstiefe sprengt den Rahmen des zur Verfügung stehenden Stacks. Es muss also ein anderer Lösungsansatz gefunden werden.

Labeling :

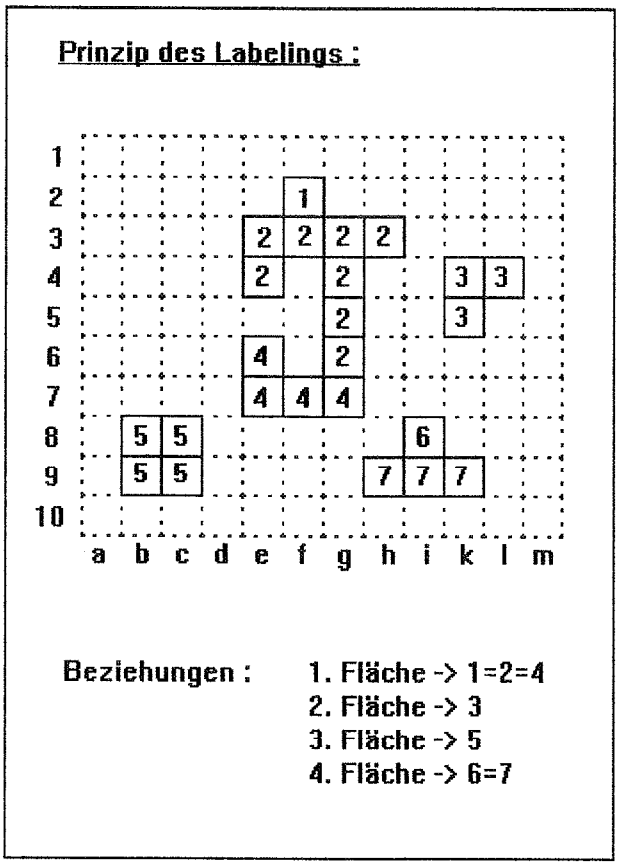


Fig. 8

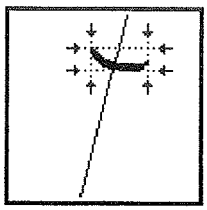
Das Bild wird von oben links nach unten rechts durchgescannt. Jedem gefundenen Pixel wird eine Nummer zugeordnet. Dabei übernimmt jedes Pixel die Nummer seines linken oder oberen Nachbarpixels. Falls links und oben kein Nachbarn ist, wird die Laufnummer um 1 inkrementiert und dem Pixel zugeordnet. Sollte es zu einem Konflikt zwischen der Nummer des linken und der Nummer des oberen Nachbarpixels kommen, so muss sich der Algoritmus merken, dass die beiden Nummern die selbe Fläche beschreiben und die linke Nummer übernehmen..

Dazu ein Beispiel anhand der Fig. 8 :

Pos X	Pos Y	i	Nachbar Links	Nachbar Oben	Definitive Pixel-Nr.	Beziehungen
2	f	1	-	-	1	
3	e	2	-	-	2	
3	f	3	2	1	2	1=2
3	g	3	2	-	2	
3	h	3	2	-	2	
4	e	3	-	2	2	
4	g	3	-	2	2	
4	k	3	-	-	3	
4	l	4	3	-	3	
5	g	4	-	2	2	
5	k	4	-	3	3	
6	e	4	-	-	4	
6	g	5	-	2	2	
7	e	5	-	4	4	
7	f	5	4	-	4	
7	g	5	4	2	4	1=2=4
8	b	5	-	-	5	
8	c	6	5	-	5	
8	i	6	-	-	6	
9	b	7	-	5	5	
9	c	7	5	5	5	
9	h	7	-	-	7	
9	i	8	7	6	7	6=7
9	k	8	7	-	7	

Jetzt kann der mittlere schwarze Flecken ausgelesen werden. Dieser wird mit dem binärisiertem Bild AND-verknüpft. Als Resultat erhält man ein "bereinigtes" Binär-Bild, das nur noch die Striche des Barcodes enthält.

Die C-Routine heisst "LargeRegionOfInterest" in IMG_PROC.H !



4.5 Balken extrahieren und eingrenzen

Jetzt muss die Richtung der Striche im Barcode erkannt werden. Dazu wird ein beliebiger Balken ausgewählt, indem von unten rechts nach oben links gesucht wird. C-Routine : "SmallRegionOfInterest" in IMG_PROC.H !

Da die Grösse dieses Balkens kleiner als 2100 Pixel ist, wird er mit einem Rekursionsalgoritmus ausgelesen. Die Rekursionstiefe übersteigt dabei die zur Verfügung stehende Stackgrösse nicht. Sollte er trotzdem einmal, wird die Rekursion abgebrochen und eine Fehlermeldung ausgegeben.

Der ausgelesene Balken wird nun eingegrenzt. Dabei sollen Eckpunkte des Balkens gefunden werden, um Mittellinie zu bilden. Ortogonal zu dieser Mittellinie zwischen den Eckpunkten A und B wird die Lesegerade zu liegen kommen.

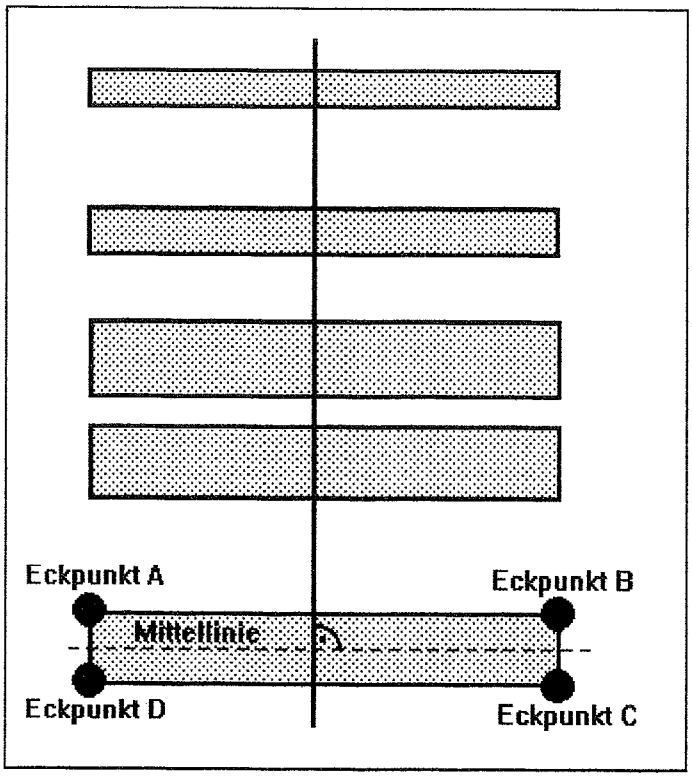


Fig. 9

Nun zeigte sich aber am vorhandenen Bildmaterial, dass ein Balken wie er in Fig. 9 dargestellt ist, reines Wunschdenken ist. Die extrahierten Balken stehen entweder schief im Bild sind gekrümmt oder haben "Nadeln" (Fig. 10, Fallbesipiel H) die das Finden von Eckpunkten zusätzlich erschweren. Darum muss das Bild auf acht verschiedene Arten durchscannt werden, (siehe Fig. 10 : Scan-Richtungen 1 bis 8) damit alle Fälle abgedeckt werden.

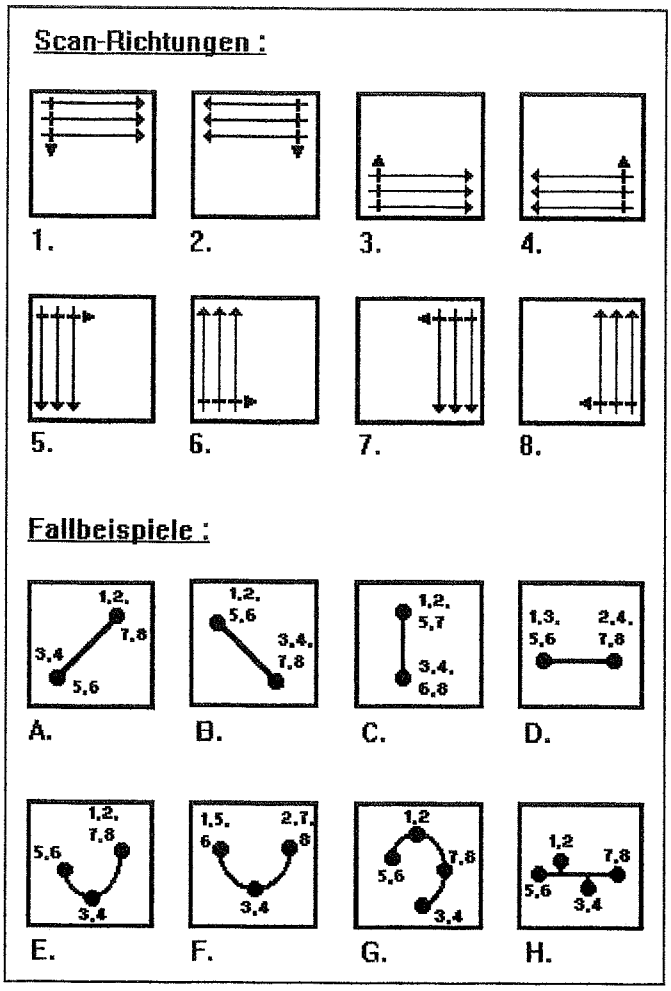


Fig. 10

Die 8 erhaltenen Punkte ergeben mit einer linearen Regression (oder lineare Ausgleichsrechnung) Lit. [D] die Mittellinie.
Bei der linearen Regression wird eine Gerade durch die gegebenen Punkte gelegt, so dass die Summe der Abstände von den Punkten zu den Geraden minimal wird. Geradengleichung : y=ax+b

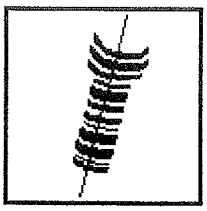
Die Koeffizienten a und b der Geraden erhält man :

$$a = \frac{n \cdot \sum x_k \cdot y_k - \sum x_k \cdot \sum y_k}{n \cdot \sum x_k^2 - (\sum y_k)^2}$$

$$b = \frac{\sum y_k - a \cdot \sum x_k}{n}$$

wobei n = Anzahl Punkte; (xk/yk) = Punktkoordinaten

Die C-Routine für das Balken eingrenzen : "SearchEdge" in IMG_PROC.H !
Die C-Routine für die Lineare Regression : "LinRegression" in BC_MATH.H !



4.6 Vektorisieren und BC-Nr. lesen

Die Lesegerade soll sich zwischen den äussersten Punkten (in Fig. 11 P1 und P7) befinden. Dazu wird von jedem Punkt (P1 - P7) der Fusspunkt auf der Mittellinie gebildet (F1 - F7). Danach werden die Abstände zwischen den Fusspunkten berechnet. Zwischen den zwei am weitesten auseinanderliegenden Fusspunkten wird die Mittelsenkrechte berechnet.

M = (F1 + F7) / 2 + F1

al = Steigung der Lesegerade
bl = Achsenabschnitt der Lesegerade
am = Steigung der Mittellinie
bm = Achsenabschnitt der Mittellinie

al = -1/am

bl = My - al * Mx

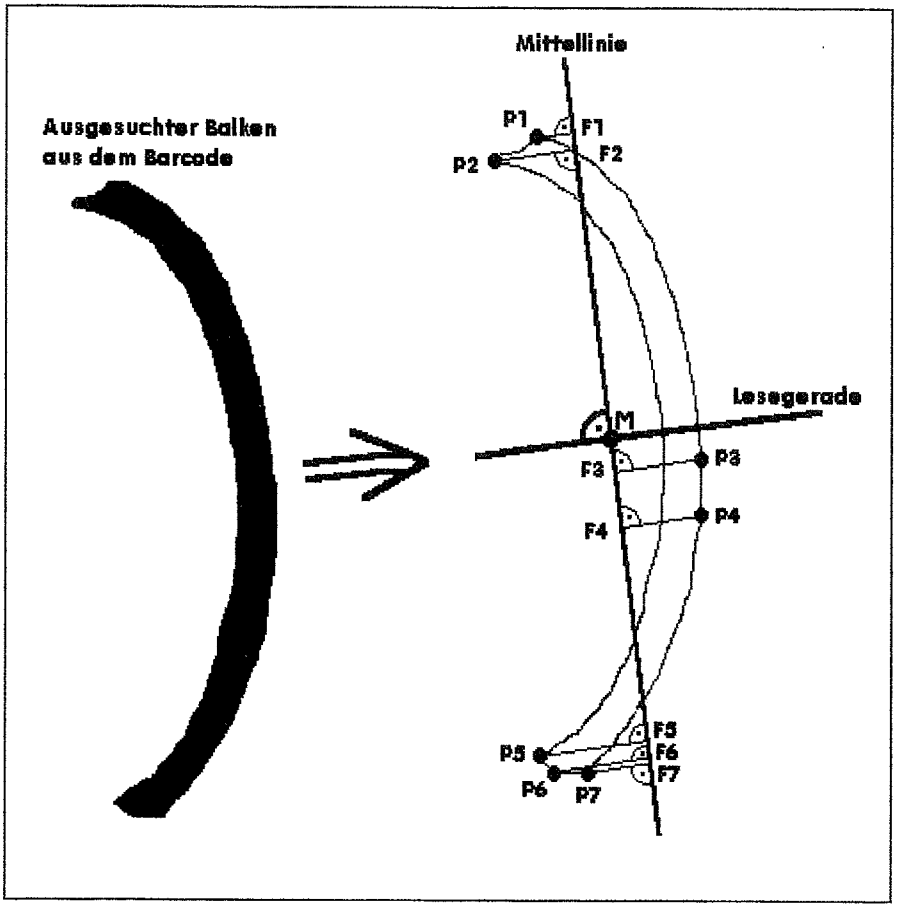


Fig. 11

C-Routine "BCGerade" in BC_MATH.H !

Nachdem die Lesegerade gefunden worden ist, geht es darum, den Code zu knacken. C-Routine "GetBarcodeNr" in BC_PROC.H !
Fig. 12 zeigt die Codabar-Codevorschrift.

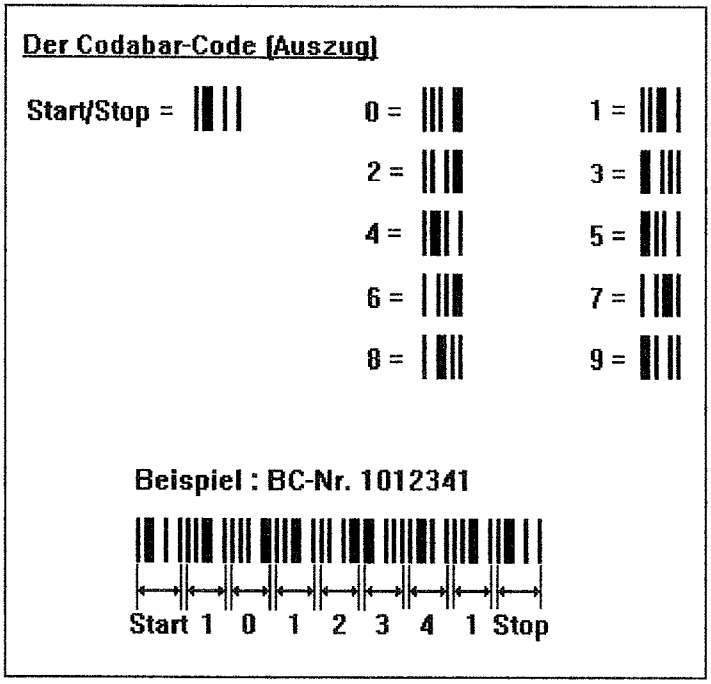


Fig. 12

Der Lesegerade entlang (siehe Fig. 13) wird der Begin des Codes gesucht. Dieser ist das Start/Stop-Zeichen. Wird das Start/Stop-Zeichen in Vorwärtsrichtung erkannt, so werden die folgenden Zeichen auch in Vorwärtsrichtung interpretiert. In Fig. 13 trifft die Lesegerade auf das Start/Stopzeichen in Rückwärtsrichtung. Somit werden die nachfolgenden Zeichen rückwärts gelesen. Der Zahlenstring muss in diesem Fall von hinten nach vorne ausgelesen werden.

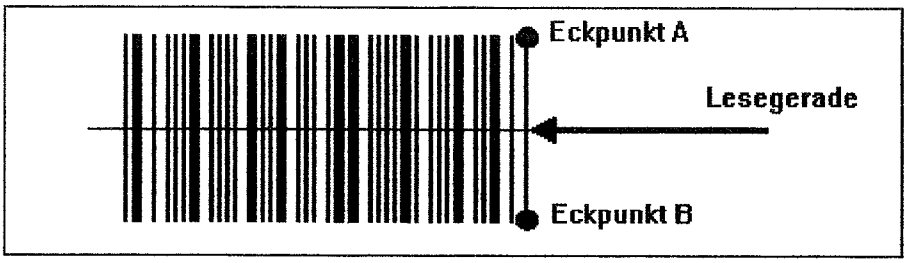


Fig. 13

Kann einer Strichabfolge kein Zeichen zugeordnet werden, muss angenommen werden, das die Lesegerade über einer Fehlerstelle liegt. Die Lesung wird abgebrochen. Für die Bildung einer neuen Lesegerade wird randommässig ein Punkt zwischen Eckpunkt A und Eckpunkt B bestimmt. Die Lesegerade wird parallel so verschoben, dass sie den Punkt schneidet. Dies kann 10 Mal wiederholt werden, bis der Barcode mit der Fehlermeldung "Unleserlich" zurückgewiesen wird.

5. Test/Besprechung

Das Programm funktioniert und erkennt 55% der bereitgestellten BC-Bilder. Bei den nicht erkannten Bilder sind entweder die Codes auch von Auge unleserlich, nicht im Bildzentrum oder beim Expandieren verläuft der Barcode-Flecken in einen angrenzende Umgebungs-Flecken, was zu Fehlermeldungen führt.

Besprechung der einzelnen Bilder (Die Bilder sind im Anhang A zu finden) :

BC022_1.TIF	(Fig. 1)	:	Schlechte Binärisierung, der Barcode verläuft hell (oben) bis dunkel (unten) und so kann keine geeignete Schwelle zum Binärisieren gefunden werden. Kein sauberes Binärisieren ergibt Codefragmente, so, dass die weiteren Verarbeitungsschritte auch keine besseren Resultate erzielen können.
BC022_2.TIF	(Fig. 2)	:	Das dunkle Bild bereitet kein Problem aber der abgeschnittene Code ergibt beim Balken extrahieren und eingrenzen eine falsche Lesegerade so dass der Code nicht gelesen werden kann.
BC037_1.TIF	(Fig. 3)	:	Erfolgreich gelesen =>1001037
BC037_2.TIF	(Fig. 4)	:	Erfolgreich gelesen =>1001037
BC039_1.TIF	(Fig. 5)	:	Erfolgreich gelesen =>1001039
BC039_2.TIF	(Fig. 6)	:	Der ausgewählte Balken zur Bestimmung der Grundlinie ist wegen der Binärisierung stark fragmentiert. Der Balkeneingrenzungsalgoritmus findet nur einen kleinen Teil dieses Balkens (1 Pixel breit). Die Lesegerade hat keinen Spielraum um sich parallel zu verschieben, falls die erste Lesung fehlerhaft war. Die 10 Leseversuche müssen immer auf der selben Gerade durchgeführt werden.
BC040_1.TIF	(Fig. 7)	:	Die Lesung ist falsch weil der Schnitt eine Bitabfolge ergibt die im Codeabar nie vorkommt.
BC041_1.TIF	(Fig. 8)	:	Wegen einer schlechten Binärisierung ist das Bild verstückelt (fragmentiert) und es kann keine Segmentierung (Region of Interest) durchgeführt werden.
BC041_2.TIF	(Fig. 9)	:	Erfolgreich gelesen =>1001041
BC041_3.TIF	(Fig. 10)	:	Erfolgreich gelesen =>1001041
BC042_1.TIF	(Fig. 11)	:	Erfolgreich gelesen =>1001042
BC044_1.TIF	(Fig. 12)	:	Erfolgreich gelesen =>1001044

BC044_2.TIF	(Fig. 13) :	Die oberen Balken sind bei Betrachtung des Bildes von Auge nicht eindeutig erkenn- bzw. unterscheidbar. Somit kann das der Computer auch nicht.
BC045_1.TIF	(Fig. 14) :	Erfolgreich gelesen =>1001045
BC045_2.TIF	(Fig. 15) :	Erfolgreich gelesen =>1001045
BC045_3.TIF	(Fig. 16) :	Wegen schlechter Binärisierung und "ausgefranstem" Balken der für die Grundlinie ausgewählt wurde, kann keine saubere Lesegerade gefunden werden.
BC045_4.TIF	(Fig. 17) :	Da sich der Barcode nicht in Zentrumsnähe befindet, so wie das der entsprechende Algoritmus erwartet, wird beim "Region of Interest" der schwarze Steg (in der Mitte) ausgewählt.
BC341_1.TIF	(Fig. 18) :	Beim Expandieren verläuft der Streifen an der Randzone mit dem Barcode. Beim suchen eines Balkens setzt der Algoritmus auf den Streifen auf.
BC341_7.TIF	(Fig. 19) :	Erfolgreich gelesen =>1012341
BC341_8.TIF	(Fig. 20) :	Erfolgreich gelesen =>1012341
BC341_9.TIF	(Fig. 21) :	Erfolgreich gelesen =>1012341
BC341_a.TIF	(Fig. 22) :	Erfolgreich gelesen =>1012341
BC341_6.TIF	(Fig. 23) :	Erfolgreich gelesen =>1012341
BC341_5.TIF	(Fig. 24) :	Beim Expandieren verläuft der Streifen an der Randzone mit dem Barcode. Beim suchen eines Balkens setzt der Algoritmus auf den Streifen auf.
BC341_2.TIF	(Fig. 25) :	Erfolgreich gelesen =>1012341
BC341_3.TIF	(Fig. 26) :	Beim Expandieren verläuft der Streifen an der Randzone mit dem Barcode. Beim suchen eines Balkens setzt der Algoritmus auf den Streifen auf.
BC341_4.TIF	(Fig. 27) :	Beim Expandieren verläuft der Streifen an der Randzone mit dem Barcode. Beim suchen eines Balkens setzt der Algoritmus auf den Streifen auf.

Total Barcodes	:	27
davon erkannt	:	15
und nicht erkannt	:	12

Ein weiters Problem ist die Verarbeitungszeit von 14 Sekunden :

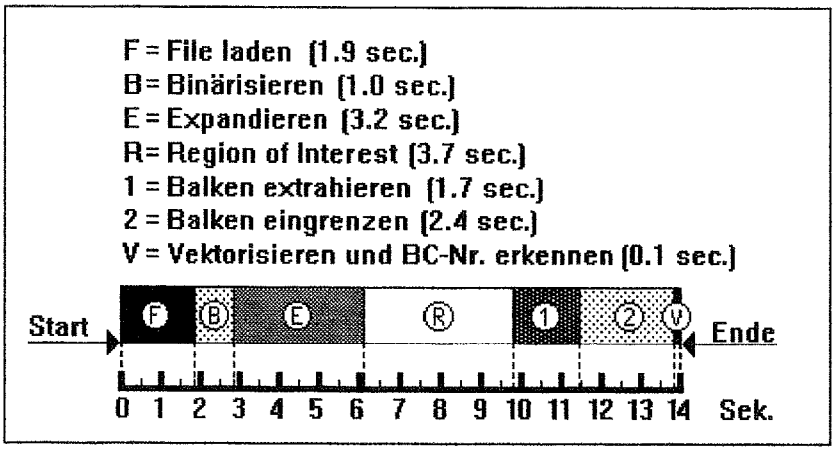


Fig. 14

Messkonfiguration

Rechnertyp	:	Intel 80486DX, VESA-Local-Bus
Systemclock	:	66MHz
Betriebssystem	:	MS-DOS V6.0
TIF-File	:	BC044_1.TIF (die anderen TIF-Files liegen zeitgleich !)

Kommentar

File laden	Nicht relevanter Anteil
Binärisieren	Unumgänglicher Verarbeitungsschritt
Expandieren	Insgesammt vier Durchgänge : 1. Rasterung, 2.- 4. Expandieren
Region of Interest	Aufwendiges Labeling (Rekursion wäre bedeutend schneller)
Balken extrahieren	Eine Rekursion und dementsprechend schnell
Balken eingrenzen	Da im voraus die Balkenform und Lage nicht bekannt ist, muss das Bild achtmal durchsucht werden
Vektorisieren und BC-Nr. erkennen	Vernachlässigbarer Zeitanteil

6. Wie weiter ?

Das Hauptproblem stellt die lange Verarbeitungszeit von 14 Sekunden dar. Wobei für alle Bilder die selbe Zeit zu veranschlagen ist, weil jedesmal dieselben aufwendigen Schritte durchgeführt werden. Zwar kann es im letzten Verarbeitungsschritt bei der Code-Interpretierung etwas differieren. Wird der Code nicht beim ertsen Mal gelesen werden, muss bekanntlich eine zweite, dritte usw. Lesung durchgeführt werden. Wie aber Fig. 14 zeigt, ist der zeitliche Anteil dieses Verarbeitungsschrittes gegenüber den übrigen Zeiten vernachlässigbar.

Zirka eine Sekunde könnte noch eingespart werden, wenn im Verarbeitungsschritt "Region-of-Interest" beim Labeling ein etwas raffinierterer Algoritmus gewählt würde.

Auch benötigen die Routinen "get_binaer" und "put_binaer" für das Lesen und Schreiben einzelner Bits eine gewisse Zeit. Wäre der Speicherbedarf kein Thema, könnte pro Bit ein Byte alloziert werden und somit auf die beiden obigen Funktionen verzichtet werden.

Ein anderer Lösungsansatz wäre, auf die Bildaufbereitung zu verzichten und unmittelbar nach dem Binärisieren mit einer Lesegeraden auf die Barcodesuche zu gehen (siehe Fig. 15) :

Dabei müssen auf der Lesegerade aus dem Zentrum heraus die Striche gelesen werden, bis man auf ein Start/Stop-Zeichen trifft. Wird kein Start/Stop-Zeichen gefunden, muss die Lesegerade um zB. 30 Grad gedreht werden und anschliessend erneut mit der Stricherkennung begonnen werden. Voraussetzung wäre aber, dass das Bild frei von störenden Umrandungen, Schatten usw. ist.

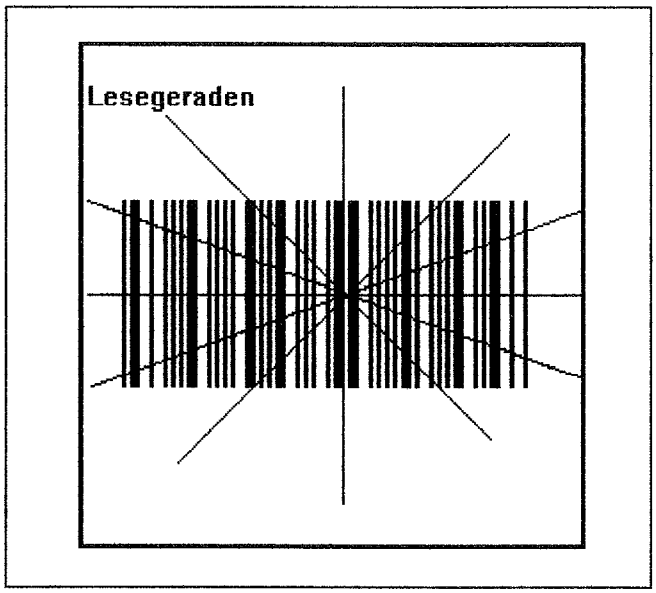


Fig. 15

Bei besseren Ausgangsbilder könnten aber auch bei der gewählten Verarbeitungsprozedur Abstriche und somit Zeiteinsparungen gemacht werden !

Auch denkbar wäre eine Bildvorverarbeitung (zB. das Binärisieren) auf spezieller HW oder der Einsatz eines Multiprozessor-Systems.

7. Programmbenutzung

Das Programm (video_bc.exe) wird folgendermassen aufgestartet :

Normalmodus : video_bc <filename>
Showmodus : video_bc <filename> s (filename zB. bc037_1.tif)

Im Showmodus wird nach jedem Verarbeitungsschritt angehalten und das neu berechnete Bild im Graphikmodus angezeigt. Die Aufforderung, eine Taste zu drücken kann gebraucht werden, um die Verarbeitung abzubrechen :

Eingabe "e" = Abbruch

Dabei erscheint die Meldung

```
VIDEO_BC.EXE : Process aborted
```

Meldungen an den Benutzer :

VIDEO BC.EXE : Successful terminated
Die Verarbeitung konnte fehlerfrei abgeschlossen werden. Die Barcodenummer wird angezeigt.

VIDEO BC.EXE : Not enough memory
Die Speicherallozierroutinen AllozieréByte oder AllozieréInt konnten nicht genügend Memory allozieren. Eventuell die Speicherbelegung überprüfen (Norton-Commander abschalten, residente Applikationen abbrechen usw. Falls immer noch zu wenig Speicher verfügbar ist, kann mit einem Speicheroptimierer wie zB. QEMM 386 (Quaterdeck Expanded Memory Manager) mehr Platz im 640k-Memory-Raum geschaffen werden.

VIDEO BC.EXE : File open error
File nicht vorhanden

VIDEO BC.EXE : Fileformat error
File entspricht nicht dem geforderten TIF-Format

VIDEO BC.EXE : Unknown file
Unbekanntes File-Format

VIDEO BC.EXE : Uninterpretable barcode
Barcode kann nach zehntem Versuch nicht gelesen werden.

```
VIDEO_BC.EXE : video_bc <filename>    = Process barcode
               video_bc <filename> s = Process and show
```
Programm falsch oder mit falschen Parametern aufgerufen.
Der richtige Aufruf sei hiermit nochmals gezeigt !

VIDEO_BC.EXE : Process aborted
Abbruch der Verarbeitung im Show-Modus. Im Normal-Modus kann die Verarbeitung durch den Benutzer nicht unter- oder abgebrochen werden.

```
VIDEO_BC.EXE : No filename !
               video_bc <filename> [s=show]
```
Filename vergessen anzugeben.

8. Literaturverzeichnis

Lit.[A] Digitale Bildverarbeitung mit dem PC
Hans-Jürgen Schlicht, Addison-Wesley
ISBN 3-89319-594-7

Lit.[B] Bildverarbeitung in der Praxis
R. Steinbrecher, Oldenbourg
ISBN 3-489-22372-0

Lit.[C] Digitale Bildverarbeitung
P. Haberäcker, Hanser
ISBN 3-446-16339-5

Lit.[D] Numerische Mathematik mit Turbo-Pascal
Ernst Kansen, Hüthig
ISBN 3-7785-1477-6

Lit.[E] Programmieren in C
Kernighan/Ritchie, Hanser
ISBN 3-446-15497-3

Lit.[F] PC intern Systemprogrammierung
Tischer, Data Becker
ISBN 3-89011-591-8

Lit.[G] Barcode Schnellkurs
Uwe Fernengel, Mister Software
ISBN 3-88963-267-X

9. Anhang : Headerfiles

```
/*********************************************************************/
/*********************************************************************/
/*** File         : SYSPARAM.H                                     ***/
/*** Beschreibung : System-Parameter                               ***/
/*********************************************************************/
/*********************************************************************/
#define ANZ_PIXEL_IN_X  512          /* Bildgroesse in x              */
#define ANZ_PIXEL_IN_Y  512          /* Bildgroesse in y              */
#define XB_OFFSET       120          /* Bildoffset  in x              */
#define YB_OFFSET         2          /* Bildoffset  in y              */
#define RASTERUNG         2          /* Rasterung (Erfahrungswert)    */
#define RASTERRUNDUNG    30          /* Rundung   (Erfahrungswert)    */
#define ANZAHLVERSUCHE    9          /* Anz. BC-Leseversuche          */

/*********************************************************************/
/*********************************************************************/
/*** File         : BC_CST.H                                       ***/
/*** Beschreibung : Konstanten fuer die Module :                   ***/
/***                    - BC_PROC.C bzw. BC_PROC.H                 ***/
/***                    - BC_SHOW.C bzw. BC_SHOW.H                 ***/
/*********************************************************************/
/*********************************************************************/
/* Fehlermeldungen bzw. Funktionsrueckgabewert aus : BC_Process()      */
/*                                                   BC_ProcessShow()  */
#define INTERPRET_BC     1
#define NOT_ENOUGH_MEM   0
#define UNINTERPRET_BC  -1
#define PROCESSABORTED  -2

/*********************************************************************/
/*********************************************************************/
/*** File         : TIF_PARS.H                                     ***/
/*** Beschreibung : Funktion zur Extrahierung                      ***/
/***                von Tags im TIF-File                           ***/
/*********************************************************************/
/*********************************************************************/
/* Fehlermeldungen aus get_picture() (Funktionsrueckgabewert) */
#define IS_TIF        1
#define IS_UNKNOWN    0
#define IS_ERROR     -1
#define FOPEN_ERROR  -2
#define NO_MEMORY    -3
#define TIF_MISMATCH -4
#define MISC_ERRORS  -5
int get_picture(char filename[], char far **fm);
```

```
/*********************************************************************/
/*********************************************************************/
/*** File         : MEMALLOC.H                                      ***/
/*** Beschreibung : Funktionen zur dynamischen Speicherallozierung  ***/
/***                und Bitextrahierung aus einem Byte              ***/
/*********************************************************************/
/*********************************************************************/
#define TRUE    1
#define FALSE   0
#define WEISS   1
#define SCHWARZ 0
/* Byte-Matrix dynamisch allozieren */
char far **  AlloziereByte(int zeilen, int spalten, int * fehlerstatus);
/* Integer-Matrix dynamisch allozieren */
unsigned int far ** AlloziereInt(int zeilen,
                                 int spalten,
                                 int * fehlerstatus);
/* Byte-Matrix deallozieren */
void  DeallozierenByte(int zeilen, char far ** fmatrix);
/* Integer-Matrix deallozieren */
void  DeallozierenInt(int zeilen, unsigned int far ** fmatrix);
/* Einzelnes Bit in ein Byte aus der Bytematrix schreiben */
void  put_binaer(unsigned char bit, int x,int y, char far ** fmatrix);
/* Einzelnes Bit aus einem Byte der Bytematrix lesen */
char  get_binaer(int x,int y, char far ** fmatrix);

/*********************************************************************/
/*********************************************************************/
/*** File         : IMG_PROC.H                                      ***/
/*** Beschreibung : Funktionen fuer die Bildverarbeitung            ***/
/*********************************************************************/
/*********************************************************************/
/* Histogramm des Grauwertbildes erstellen. hgm[0] = Anzahl Schwarztöne*/
/*                                          hgm[255] = Anz. Weisstöne  */
void Histogramm(char far ** fm, unsigned long hgm[]);
/* Schwelle, bei der Schwarz/Weiss getrennt wird bestimmen */
int CalcBinlimit(unsigned long hgm[]);
/* Grauwertbild mit der gefundenen Schwelle binaerisieren */
void Binary(int limit,        /* I:Limite bei Binaerisierung */
            char far ** fm,   /* I:Grauwert-Bild             */
            char far ** fb);  /* O:Binaerisiertes Bild       */
/* Bild groeber Rastern für schnelleres Expandieren/*
void Coarsing(int scale,       /* I:Vergroeberungs-Faktor       */
              int limit,       /* I:%-Limite bei Binaerisierung */
              char far ** fb,  /* I:Binaerisiertes Bild         */
              char far ** fc); /* O: gerastertes Bild           */
/* Binaerisiertes Bild expandieren */
void ExpandBinaer(int scale,       /* I:Rasterung des Bildes */
                  char far ** fi,  /* I:Binaerisiertes-Bild  */
                  char far ** fo); /* O:Expandiertes Bild    */
/* Expandierter Strichcode erkennen und ausschneiden */
void LargeRegionOfInterest(int scale,
                           char far ** fi,  /* I:Expandiertes Bild     */
                           char far ** fb,  /* I:Binaerisiertes Bild   */
                           char far ** fo); /* O:Strichcode extrahiert */
/* Balken extrahieren */
void SmallRegionOfInterest(char far ** fi,  /* I:Strichcode-Bild     */
                           char far ** fo); /* O:Extrah. Balken-Bild */
/* Balken in ein Rechteck eingrenzen */
void SearchEdge(unsigned int far **Edge,  /* O:Acht Eckpunkt-Paare */
                char far ** fb);          /* I:Balken-Bild         */
```

```
/*********************************************************************/
/*********************************************************************/
/*** File         : CODABAR.H                                      ***/
/*** Beschreibung : Header zu den Routinen fuer die Bearbeitung     ***/
/***                eines Pixelstrings in eine Codabar-BC-Nummer    ***/
/*********************************************************************/
/*********************************************************************/
int GetBarcodeNr(char far ** fb,        /* I:Strichcode-Bild        */
                 char far ** PixelStr, /* I:Pixelstring            */
                 int PixelStrLen,       /* I:Länge des Pixelstrings */
                 float a, float b,      /* I:Lesegerade             */
                 char BC_Nr[]);         /* O:Barcode-Nummer         */

/*********************************************************************/
/*********************************************************************/
/*** File         : BC_MATH.H                                      ***/
/*** Beschreibung : Mathematik-Funktionen                          ***/
/*********************************************************************/
/*********************************************************************/
/* Lineare Regression */
void LinRegression(unsigned int far **Edge, /* I: Die acht Eckpunkte */
                   float * a, float * b);   /* O: Mittellinie        */
/* Berechnet die Lesegerade die durch den Barcode verlaufen muss */
void BCGerade(unsigned int far **Edge, /* I:Acht Eckpunkte
              float  a1, float  b1,    /* I:Mittelinie
              float *a2, float *b2,    /* O:Lesegerade
              unsigned int *x1,        /* O:Aeusserster Eckpunkt
              unsigned int *y1,        /* O:Aeusserster Eckpunkt
              unsigned int *x2,        /* O:Aeusserster Eckpunkt
              unsigned int *y2);       /* O:Aeusserster Eckpunkt
/* Berechnen einer zur Lesegerade paralle Gerade durch P(x/y) */
void Parallele(unsigned int x,         /* I:Neuer Punkte auf m */
               unsigned int y,         /* I:Neuer Punkte auf m */
               float a, float *b);     /* I/O:Neue Lesegerade */

/*********************************************************************/
/*********************************************************************/
/*** File         : BC_PROC.H                                      ***/
/*** Beschreibung : Hauptfunktion zur BC-Bestimmung                ***/
/*********************************************************************/
/*********************************************************************/
#include "bc_cst.h"
int BC_Process(char far ** fs,  /* I:Grauwertbild   */
               char BC_Nr[]);   /* O:Barcode-Nummer */

/*********************************************************************/
/*********************************************************************/
/*** File         : BC_GRAPH.H       NUR SHOW-MODUS !!!            ***/
/*** Beschreibung : Funktionen zur graph. Darstellung der Bitmaps   ***/
/*********************************************************************/
/*********************************************************************/
enum Info { busy, keypress, fileload };
/* Grösse des darzustellenden Bildes abchecken */
int CheckSize(int x_offset, int y_offset,  /* I:Entfernung zum Rand */
              int * x_size, int * y_size); /* I:Bildgroesse         */
/* Window löschen */
void ClearWindow(int x_offset, int y_offset,  /* I:Entfernung zum Rand */
                 int x_size,   int y_size);   /* I:Bildgroesse         */
/* Rahmen um das Window setzen */
void MakeBorder(int x_offset, int y_offset, /* I:Entfernung zum Rand */
                int x_size,   int y_size,   /* I:Bildgroesse         */
                int color);                 /* I:Farbwert            */
/* Barcode-Bild zeigen, das grob gerastert wurde */
void MakeGraphCoarse(int x_offset, int y_offset, /*I:Entfernung zum Rand*/
                     int x_size,   int y_size,   /*I:Bildgroesse        */
                     int scale,                  /*I:Rasterung          */
                     char far **fm);             /*O:Gerastertes Bild   */
/* Barcode-Bild zeigen, das binaerisiert wurde */
void MakeGraphBin(int x_offset, int y_offset, /* I:Entfernung zum Rand */
                  int x_size,   int y_size,   /* I:Bildgroesse         */
                  char far **fm);             /* I:Binaerisiertes Bild */
```

```
/* Ausgangsbild mit Grauwerten anzeigen */
void MakeGraphGray(int x_offset, int y_offset, /* I:Entfernung zum Rand */
                   int x_size,   int y_size,   /* I:Bildgroesse         */
                   char far **fm);             /* I:Grauwertbild        */
/* Berechnetes Histogramm mit eingezeichnetem Schwellwert anzeigen */
void MakeHisto(int  x_offset, int y_offset, /* I:Entfernung zum Rand */
               int  x_size,                 /* I:Bildgroesse         */
               unsigned long hgm[256],      /* I:Pixel pro Grauwert  */
               int  binlimit);              /* I:Binaer. Schwelle    */
/* Mittellinie oder Leselinie anzeigen */
void ShowLine(int x_offset, int y_offset, /* I:Entfernung zum Rand */
              int  x_size,   int  y_size, /* I:Bildgroesse         */
              float a,       float b,     /* I:Gerade              */
              int color);                 /* I:Farbwert            */
/* Eckpunkte mit Zeigern markieren */
void ShowEdge(int x_offset, int y_offset, /* I:Entfernung zum Rand */
              unsigned int far **Edge);   /* I:Acht Eckpunkte      */

/* Barcodenummer im Graphikmodus anzeigen */
void ShowBarcode(int x_offset, int y_offset, /* I:Entfernung zum Rand */
                 int x_size,   int y_size,   /* I:Bildgroesse         */
                 char BC_Nr[]);              /* I:Barcode-Nummer      */
/* Fehlermeldung an den Benutzer */
void ShowProzessFehler(int ProzessFehler);
/* Benutzermeldungen im Textmodus */
void ShowInfoGraphModus(int InfoText);
/* Benutzermeldungen im Graphikmodus */
void ShowInfoTextModus(int InfoText);
/* VGA-Treiber installieren */
int install_graph(void);
/* VGA-Treiber deinstallieren */
void deinstall_graph(void);

/*********************************************************************/
/*********************************************************************/
/*** File         : BC_SHOW.H            NUR SHOW-MODUS !!!         ***/
/*** Beschreibung : Hauptfunktion zur BC-Bestimmung und Darstellung  ***/
/***                der Zwischenbilder                               ***/
/*********************************************************************/
/*********************************************************************/
#include "bc_cst.h"
int BC_ProcessShow(char far ** fs, int xb_size, int yb_size,
                   char BC_Nr[]);
```

10. Anhang A (TIF-Bilder)

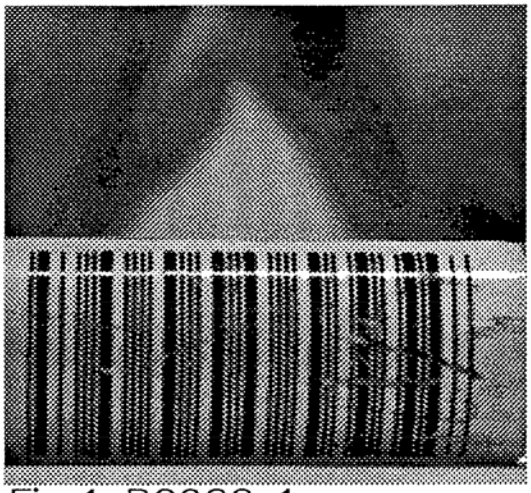


Fig.1 BC022_1

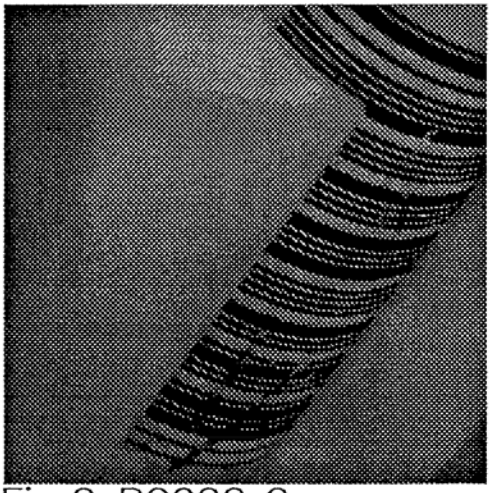


Fig.2 BC022_2

Fig.3 BC037_1

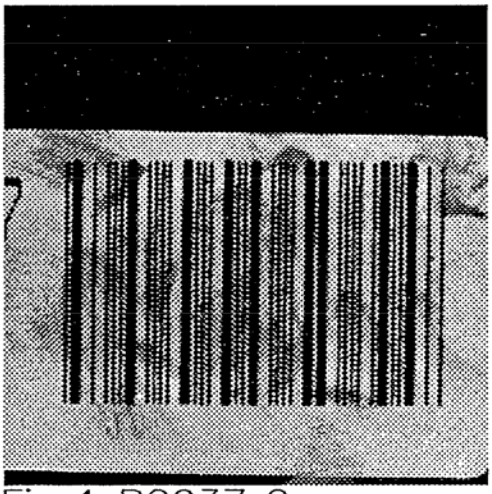


Fig.4 BC037_2

Fig.5 BC039_1

Fig.6 BC039_2

Projekt : Video Barcode Leser
Name : Jürg Arnold
Anhang A (TIF File Bilder der Barcodes)
Seite 1

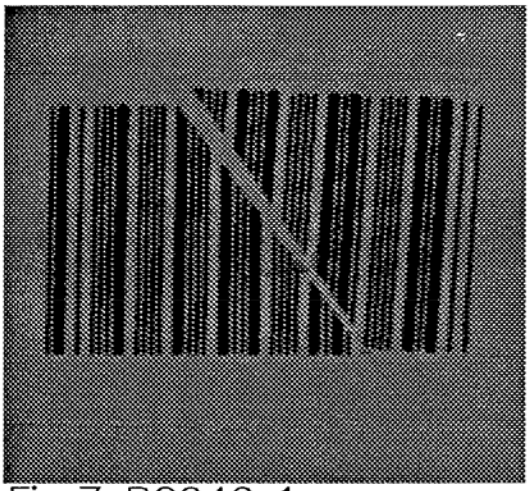


Fig.7 BC040_1

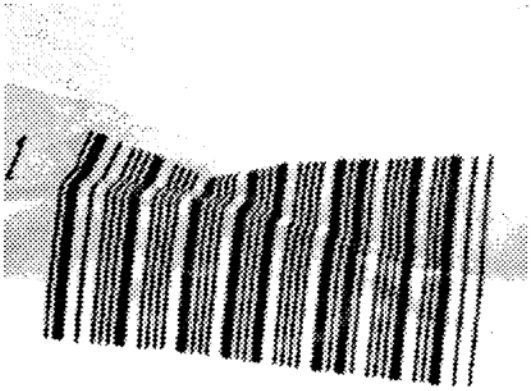


Fig.8 BC041_1

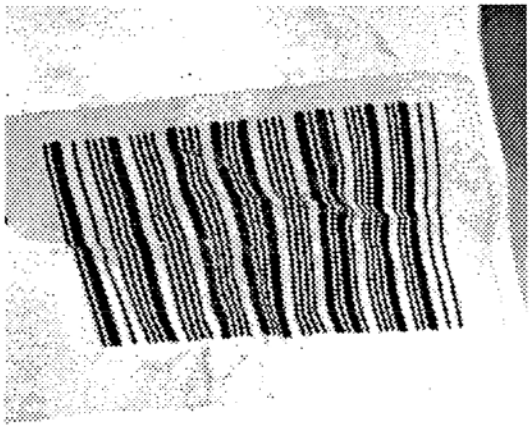


Fig.9 BC041_2

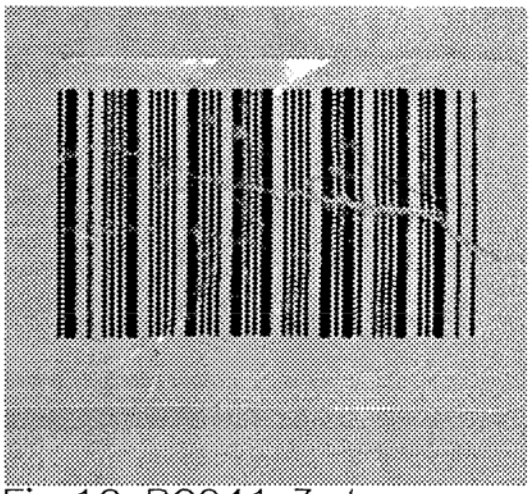


Fig.10 BC041_3

Fig.11 BC042_1

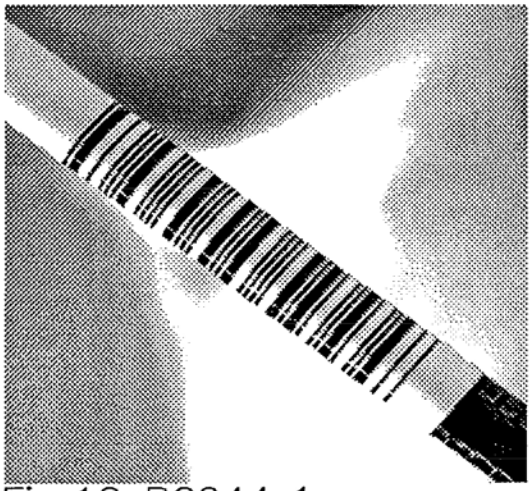


Fig.12 BC044_1

Projekt : Video Barcode Leser
Name : Jürg Arnold
Anhang A (TIF File Bilder der Barcodes)
Seite 2

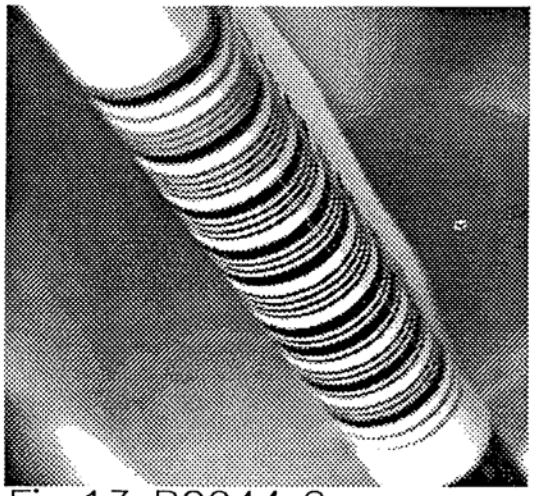


Fig.13 BC044_2

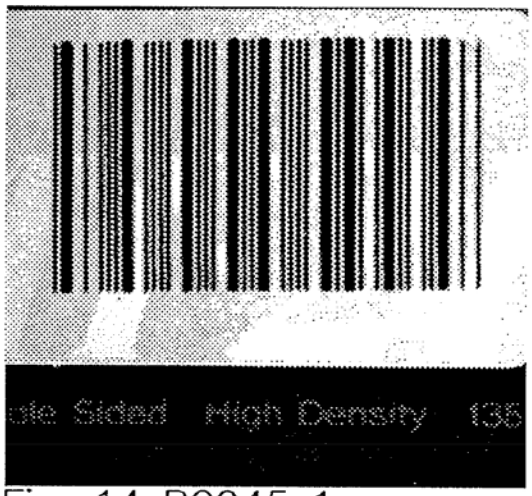


Fig. 14 BC045_1

Fig.15 BC045_2

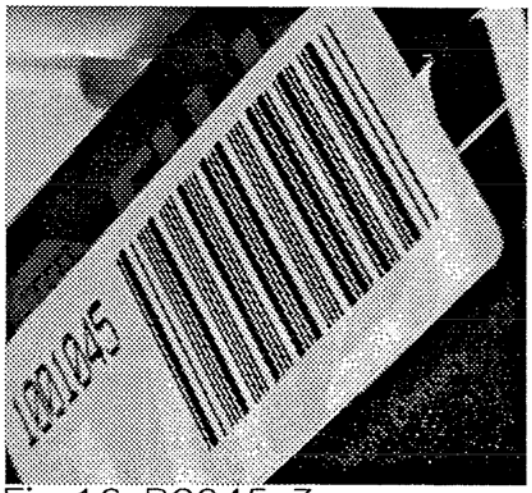


Fig.16 BC045_3

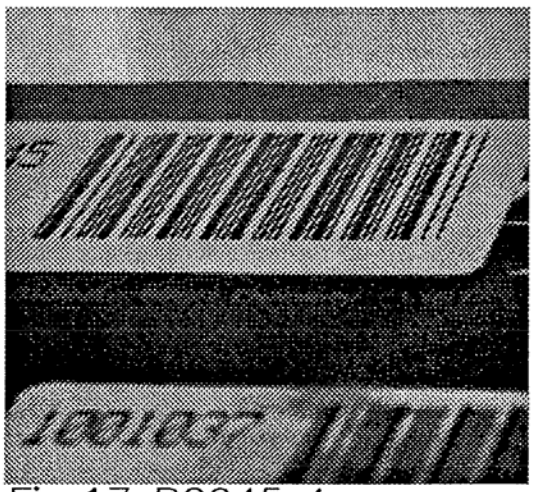


Fig.17 BC045_4

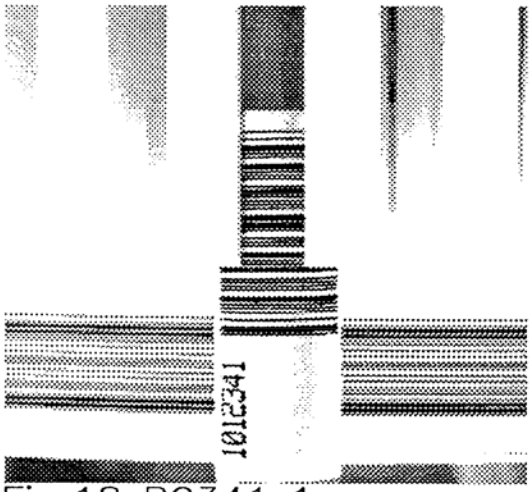


Fig.18 BC341_1

Projekt : Video Barcode Leser
Name : Jürg Arnold
Anhang A (TIF File Bilder der Barcodes)
Seite 3

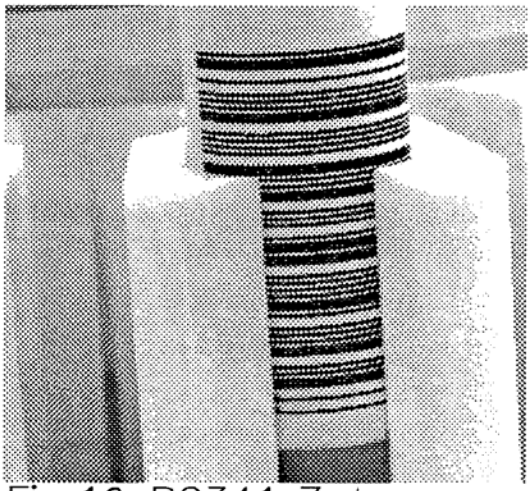


Fig. 19 BC341_7 ✓

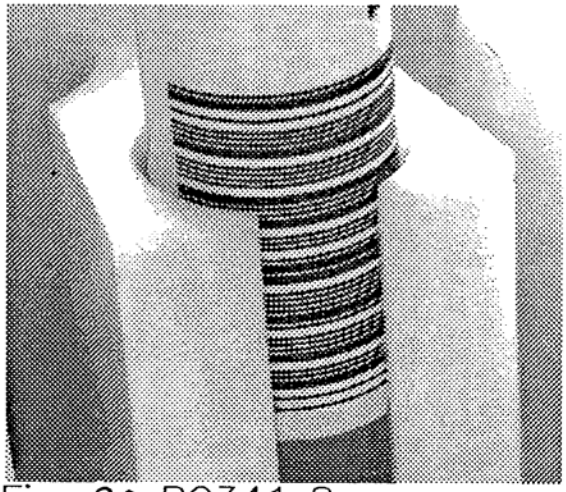


Fig. 20 BC341_8 ✓

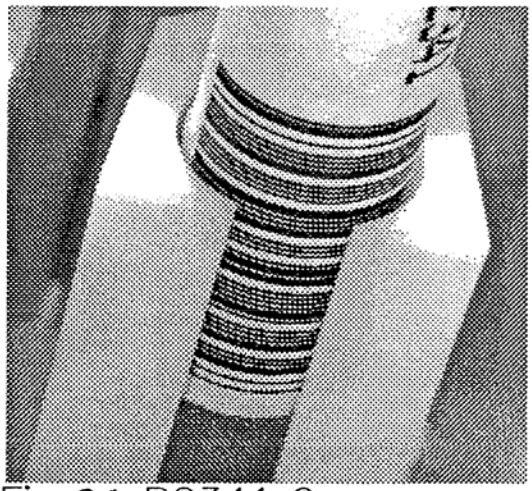


Fig. 21 BC341_9 ✓

Fig. 22 BC341_a ✓

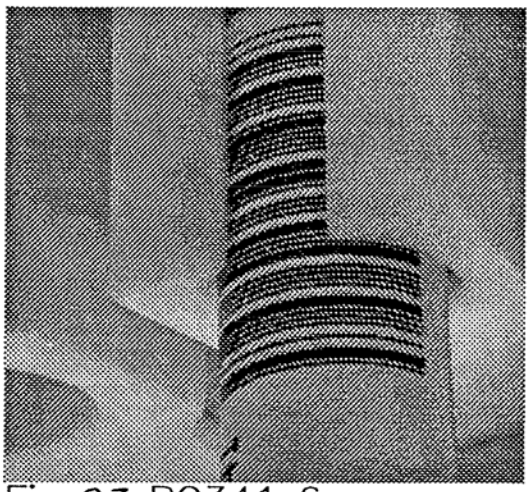


Fig. 23 BC341_6 ✓

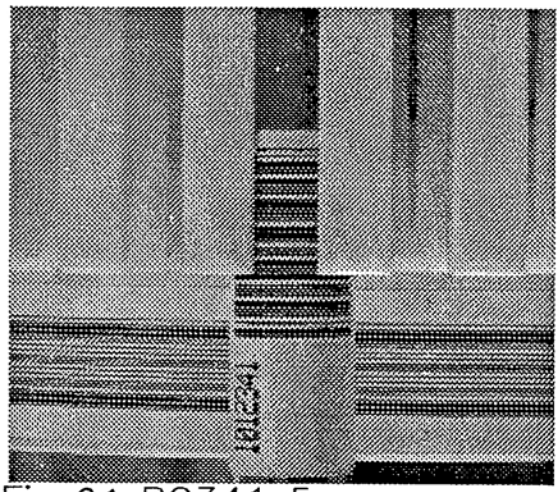


Fig. 24 BC341_5 o

Projekt : Video Barcode Leser
Name : Jürg Arnold
Anhang A (TIF File Bilder der Barcodes)
Seite 4

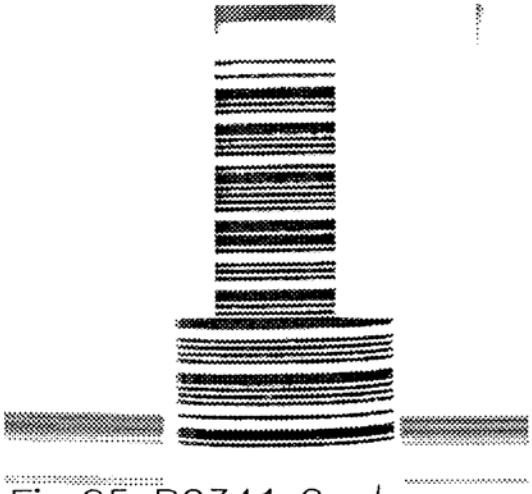


Fig.25 BC341_2

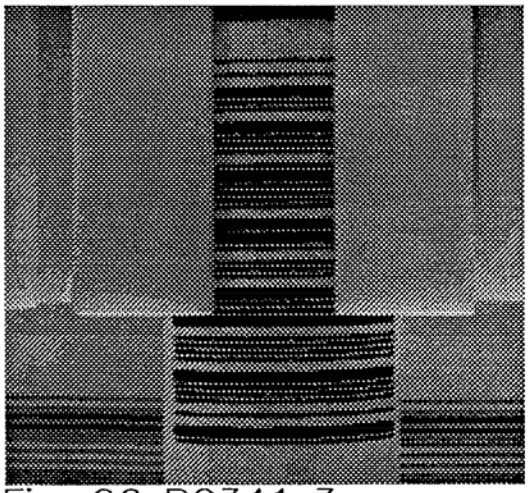


Fig. 26 BC341_3

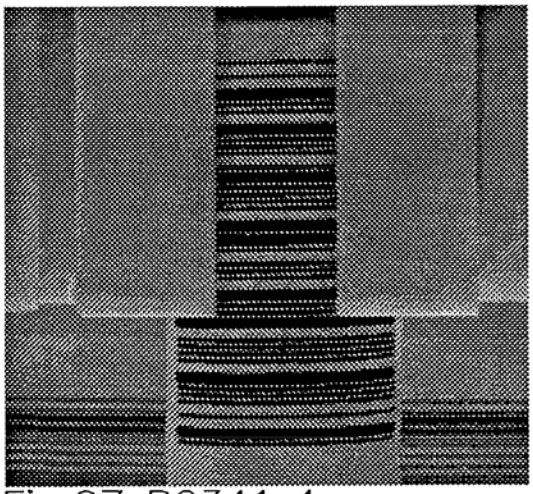


Fig.27 BC341_4