Automatische Ballerkennung

Sebastian Schütz

7. Juli 2008

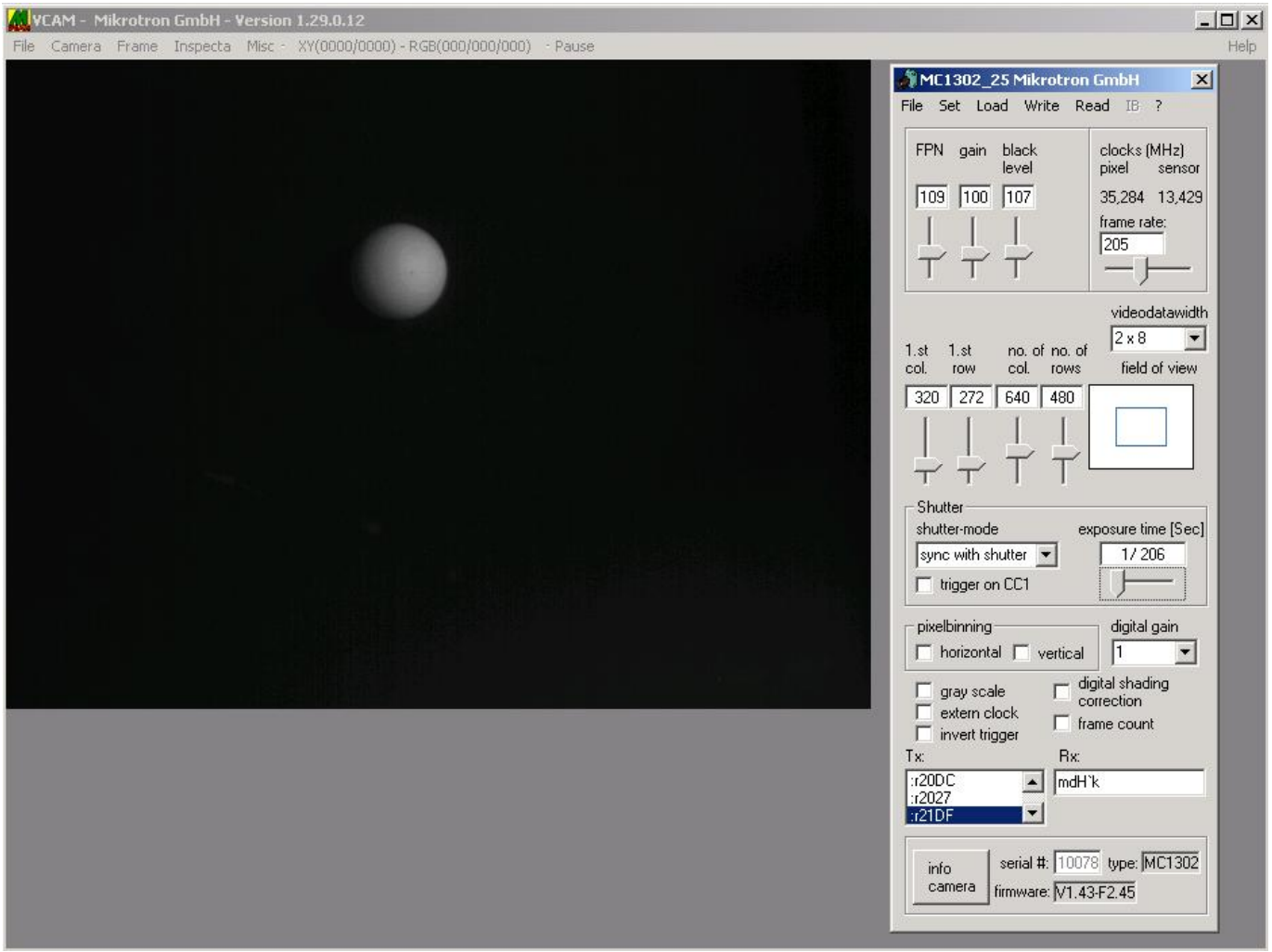
1. Kamera
   - Schrittweise Durchführung
   - Zugriff auf die Kamera
   - Aufbau von Bitmaps
   - Ballerkennung
     - Grundlegende Vorgehensweise
2. Lichtgitter
   - Einführung

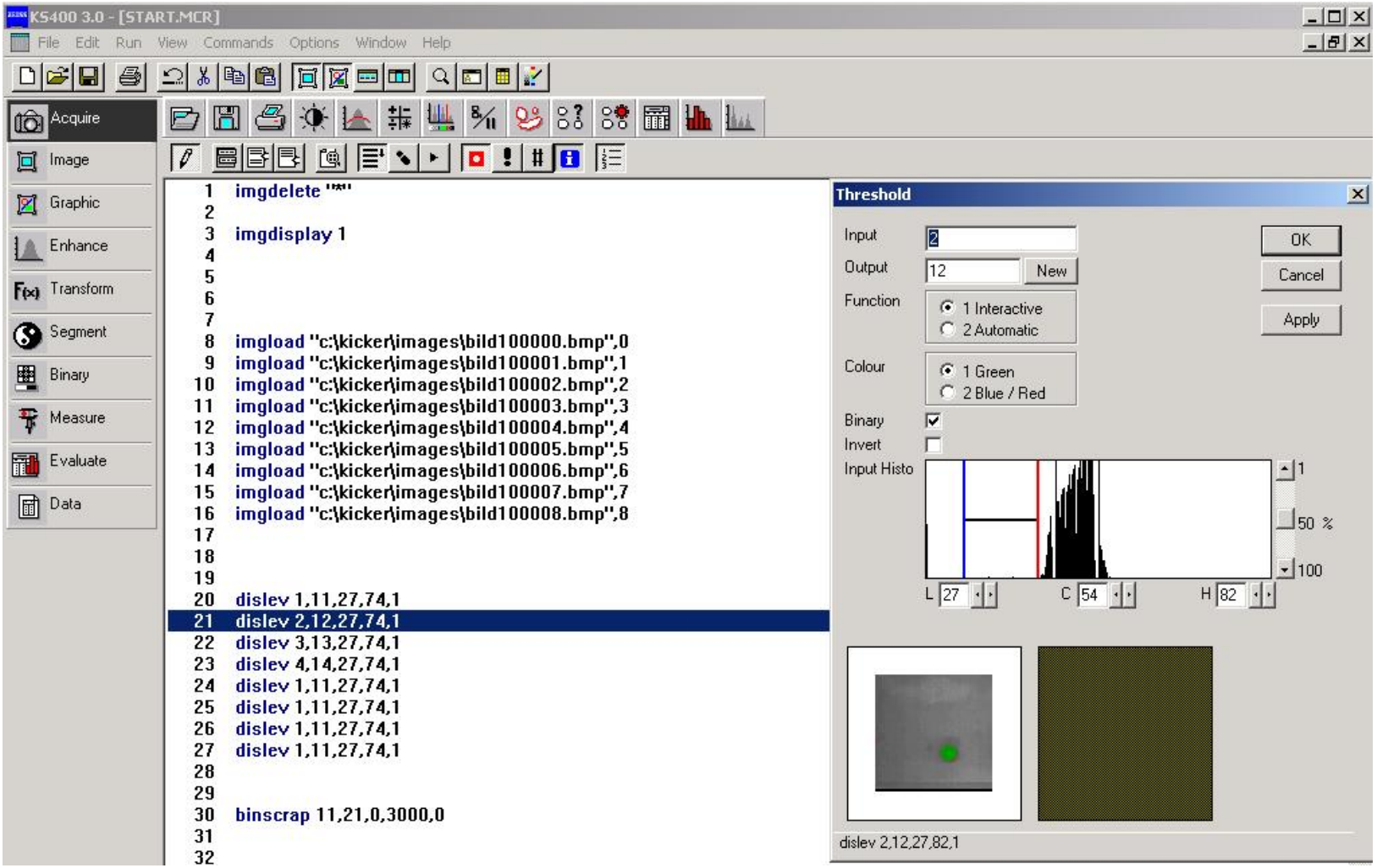
- VCAM von Mikrotron
  Kameraprofil kann geladen werden, Bilder werden angezeigt und können gespeichert werden

- VCAM von Mikrotron
  Kameraprofil kann geladen werden, Bilder werden angezeigt und können gespeichert werden
- Mikrotron camera control tool - MC1302_25 Version 1.2.9
  Das in VCAM geladene Profil kann angepasst werden (fps, exposure time, ...)

- VCAM von Mikrotron
  Kameraprofil kann geladen werden, Bilder werden angezeigt und können gespeichert werden
- Mikrotron camera control tool - MC1302_25 Version 1.2.9
  Das in VCAM geladene Profil kann angepasst werden (fps, exposure time, ...)
- KS400 von Carl Zeiss Vission GmbH
  Bildmittelpunkt von zuvor aufgenommenen bmps ermitteln (Makro)

VCAM


VCAM - Mikrotron GmbH - Version 1.29.0.12
File Camera Frame Inspecta Misc - XY(0000/0000) - RGB(000/000/000) - Pause
Help
MC1302_25 Mikrotron GmbH
File Set Load Write Read IB ?
FPN gain black level
109 100 107
clocks (MHz) pixel sensor
35,284 13,429
frame rate:
205
videodatawidth
2 x 8
1.st col. 1.st row no. of col. no. of rows
field of view
320 272 640 480
Shutter
shutter-mode
sync with shutter
trigger on CC1
exposure time [Sec]
1/ 206
pixelbinning
horizontal vertical
digital gain
1
gray scale
extern clock
invert trigger
digital shading correction
frame count
Tx:
Rx:
:r20DC
:r2027
:r21DF
mdH`k
info camera
serial #: 10078 type: MC1302
firmware: V1.43-F2.45


KS400 3.0 - [START.MCR]
File Edit Run View Commands Options Window Help
Acquire
Image
Graphic
Enhance
Transform
Segment
Binary
Measure
Evaluate
Data
1 imgdelete "*"
3 imgdisplay 1
8 imgload "c:\kicker\images\bild100000.bmp",0
9 imgload "c:\kicker\images\bild100001.bmp",1
10 imgload "c:\kicker\images\bild100002.bmp",2
11 imgload "c:\kicker\images\bild100003.bmp",3
12 imgload "c:\kicker\images\bild100004.bmp",4
13 imgload "c:\kicker\images\bild100005.bmp",5
14 imgload "c:\kicker\images\bild100006.bmp",6
15 imgload "c:\kicker\images\bild100007.bmp",7
16 imgload "c:\kicker\images\bild100008.bmp",8
20 dislev 1,11,27,74,1
21 dislev 2,12,27,74,1
22 dislev 3,13,27,74,1
23 dislev 4,14,27,74,1
24 dislev 1,11,27,74,1
25 dislev 1,11,27,74,1
26 dislev 1,11,27,74,1
27 dislev 1,11,27,74,1
30 binscrap 11,21,0,3000,0
Threshold
Input 2
Output 12
New
Function
1 Interactive
2 Automatic
Colour
1 Green
2 Blue / Red
Binary
Invert
Input Histo
OK
Cancel
Apply
1
50 %
100
L 27
C 54
H 82
dislev 2,12,27,82,1

Geschwindigkeit

Für die von uns verwendete Auflösung von $640 \cdot 480$ sollte die Kamera eine Geschwindigkeit von 206fps unterstützen. Bei unseren Tests mit VCAM konnte jedoch nur eine maximale Geschwindigkeit von ca. 49fps erzielt werden (mindestens 100fps haben wir uns als Vorgabe gesetzt).

Geschwindigkeit

Für die von uns verwendete Auflösung von $640 \cdot 480$ sollte die Kamera eine Geschwindigkeit von 206fps unterstützen. Bei unseren Tests mit VCAM konnte jedoch nur eine maximale Geschwindigkeit von ca. 49fps erzielt werden (mindestens 100fps haben wir uns als Vorgabe gesetzt).

- evtl. Festplatte zu langsam, RAM Disk könnte Abhilfe schaffen

Geschwindigkeit

Für die von uns verwendete Auflösung von $640 \cdot 480$ sollte die Kamera eine Geschwindigkeit von 206fps unterstützen. Bei unseren Tests mit VCAM konnte jedoch nur eine maximale Geschwindigkeit von ca. 49fps erzielt werden (mindestens 100fps haben wir uns als Vorgabe gesetzt).

- evtl. Festplatte zu langsam, RAM Disk könnte Abhilfe schaffen
- Ramdisk hat die Werte nicht verbessert, evtl. VCAM Schuld

Geschwindigkeit

Für die von uns verwendete Auflösung von $640 \cdot 480$ sollte die Kamera eine Geschwindigkeit von 206fps unterstützen. Bei unseren Tests mit VCAM konnte jedoch nur eine maximale Geschwindigkeit von ca. 49fps erzielt werden (mindestens 100fps haben wir uns als Vorgabe gesetzt).

- evtl. Festplatte zu langsam, RAM Disk könnte Abhilfe schaffen
- Ramdisk hat die Werte nicht verbessert, evtl. VCAM Schuld
- Abhilfe könnte evtl. durch eine eigene Implementierung realisiert werden

Geschwindigkeit

Für die von uns verwendete Auflösung von $640 \cdot 480$ sollte die Kamera eine Geschwindigkeit von 206fps unterstützen. Bei unseren Tests mit VCAM konnte jedoch nur eine maximale Geschwindigkeit von ca. 49fps erzielt werden (mindestens 100fps haben wir uns als Vorgabe gesetzt).

- evtl. Festplatte zu langsam, RAM Disk könnte Abhilfe schaffen
- Ramdisk hat die Werte nicht verbessert, evtl. VCAM Schuld
- Abhilfe könnte evtl. durch eine eigene Implementierung realisiert werden

Eigene Lösung kommt auf 100fps bei eingestellten 206fps, genaue Ursache bleibt jedoch ungeklärt

Kamera

Die von uns verwendete Kamera (MC1302 von Mikrotron) kann mit Hilfe einiger APIs gesteuert werden. Da wir die Konfiguration der Kamera mit den bereits vorhandenen Tools vornehmen können, benötigen wir nur einen kleinen Teil der zur Verfügung stehenden Funktionen.

- LONG WINAPI mvfg_open(char *pcCameraProfile, LONG DeviceNumber)
  Ermöglicht das Laden eines Kameraprofils und initialisiert das entsprechende Gerät
- LONG WINAPI mvfg_getparam(char * pcParamName, void * pValueBuffer, LONG DeviceNumber)
  Liest die momentanen Einstellungen aus der Kamera aus
- LONG WINAPI mvfg_grab(DWORD iCommand, LONG DeviceNumber)
  Startet die Belichtung und ein Bild wird in den internen Bild-Puffer geladen
- void * WINAPI mvfg_getbufptr(LONG DeviceNumber)
  Liefert den Anfang des Puffers zurück, in dem die Bilddaten gespeichert sind
- LONG WINAPI mvfg_close(LONG DeviceNumber)
  Schliesst das zuvor geöffnete Kamera-Handle
- LONG WINAPI mvfg_errmessage(LONG iCode)
  Gibt eine Message-Box mit der Fehlerbeschreibung aus (nützlich zum debuggen)

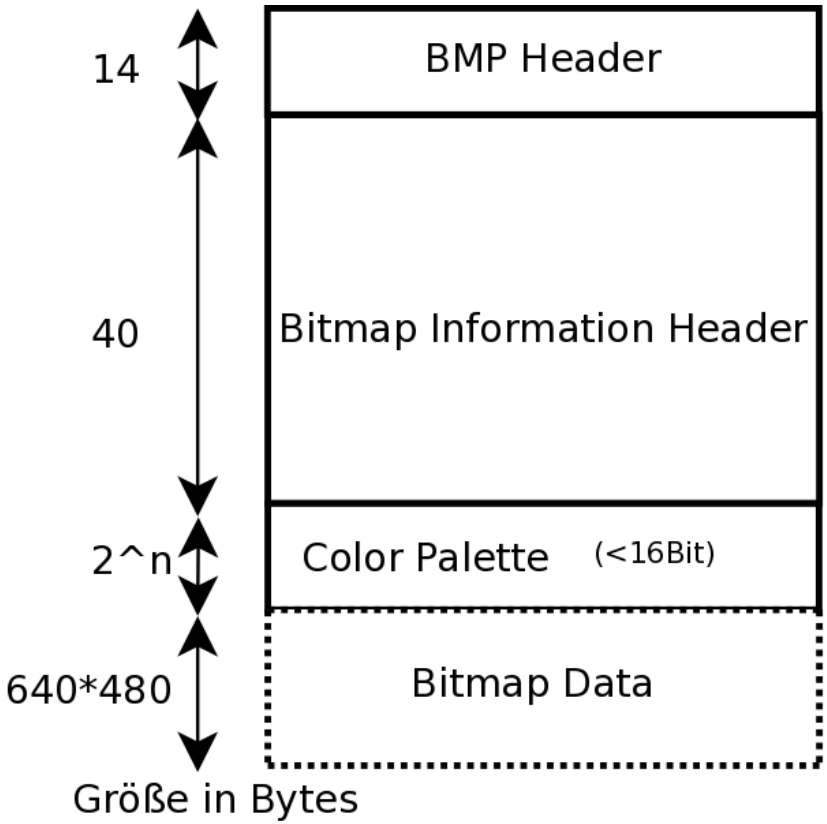
Die Funktionsprototypen befinden sich in *mvfgdrv.h*. Die Funktionen selbst müssen beim Linken mit Hilfe der *mvfgd32.lib* Library eingebunden werden.

Quellcode Ausschnitt

Folgender Code Schnipsel zeigt wie mit Hilfe der APIs ein Bild von der Kamera erfasst werden kann. Im nächsten Schritt müssen die Daten nun in eine Bilddatei gespeichert werden → *Bitmap*.

```
/* open camera with predefined profile, number 0 */
ret = mvfg_open("C:\\PROGRA~1\\MIKROT~1\\Inspecta\\Inspecta-4DC.cam;"
                "MC1310_11_02_03 640x480 206fps", 0);
/* error check */
/* get some camera informations */
ret = mvfg_getparam(MVFGPAR_FORMAT_INFO,&info,0);
/* error check */
wsprintf(buffer, "Pixel breite: %d\nPixel hoehe: %d", info.lImageWidth,
        info.lImageHeight);
MessageBox(NULL, buffer, "Informationen ueber aktuelle Einstellungen", MB_OK);
/* grab picture */
ret = mvfg_grab(GRAB_NOWAIT, 0);
if (ret != MVFG_GRAB_READY) {
    ++err;
} else {
    /* copy picture in our preallocated buffer */
    memcpy(ptr[i], mvfg_getbufptr(0), info.lFrameSize);
}
/* ... */
ret = mvfg_close(0);
```

Allgemeine Struktur eines Bitmaps

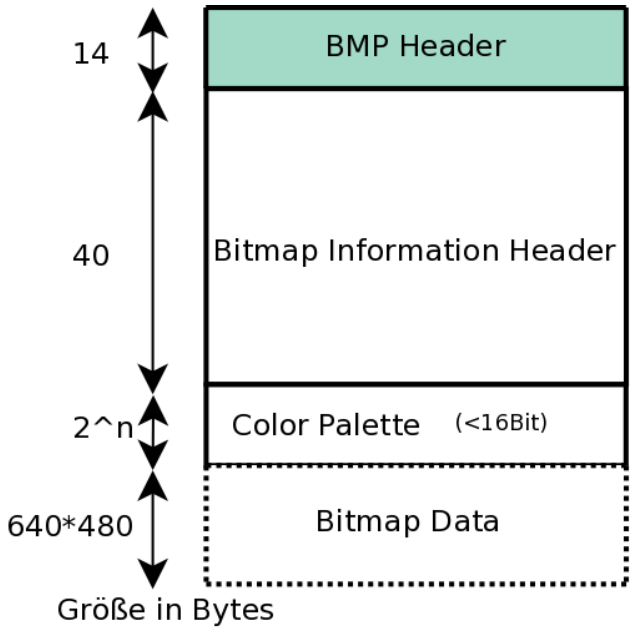


Bitmaps

Bitmaps sind sehr einfach aufgebaut und relativ schnell zu erzeugen. Neben ein paar Zusatzinformationen können direkt die Daten, die von der Kamera geliefert werden, als Bitmap abgespeichert werden. Dadurch, dass wir nur eine geringe Auflösung von 640x480 Pixel benötigen ist die Grösse des Bitmaps ungefähr bei 300kByte (inklusive Zusatzinformationen) und es ist daher nicht notwendig ein anderes Verfahren (evtl. mit Kompression, PNG, JPEG etc.) zu verwenden.

BMP Header

Durch den BMP Header wird festgelegt, dass die folgenden Daten sich auf ein Bitmap beziehen. Entscheidend hierfür ist die *Magic Number* die mit *BM* (in ASCII) angegeben wird. Zusätzlich wird ein *offset* definiert, der angibt wo die Roh-Daten des Bildes beginnen. Der Offset berechnet sich wie folgt:
$2^n \cdot sizeof(bmpRGBQuad_t) + sizeof(bmpInfoHeader_t) + sizeof(bmpFileHeader_t)$
wobei in unserem Fall $n = 8$ gilt.

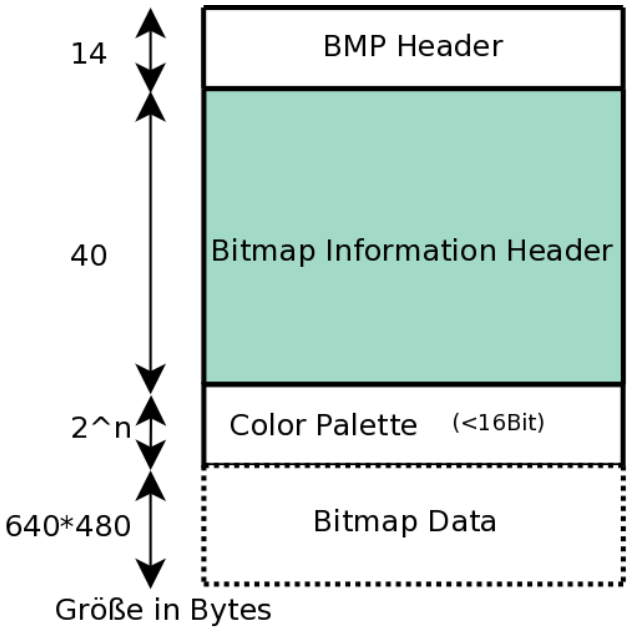


BMP Header

```
typedef struct {
   short type; //Magic Number 'BM'
   unsigned long size; //should be 0
   short res0, res1; //should be 0
   unsigned long offset;//start of data
} bmpFileHeader_t;
```

BMP Information Header

Der Bitmap Information Header gibt Auskunft über die Daten, die sich im Datenteil befinden. Als wichtigste Elemente spezifiziert er die Grösse des Bildes (*width*, *height*) und die Anzahl der verschiedenen Farben die im Bild vorkommen können.

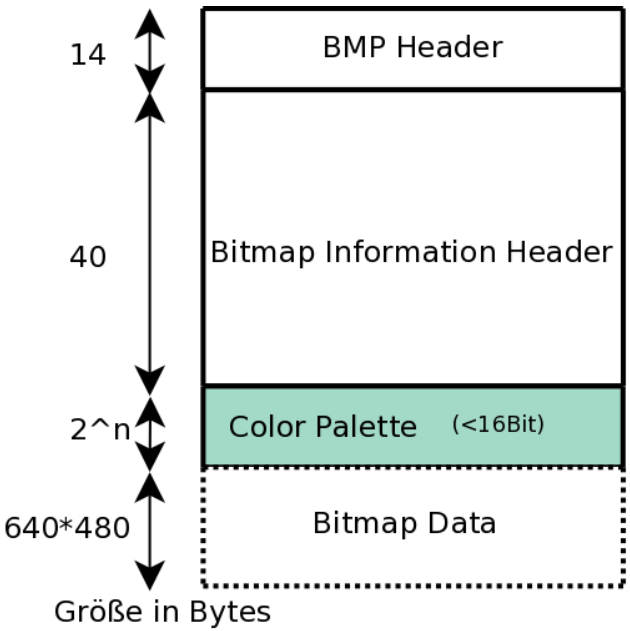


BMP Information Header

```
typedef struct {
  unsigned long size; //sizeof(bmpInfoHeader_t)
  long width, height; //640, 480
  short planes; //color planes, must be set to 1
  short cDepth; //color depth 1,4,8,16,24,32
  unsigned long compression; //set to 0
  unsigned long rawSize; //width * height
  long hRes, vRes; //both set to 0
  unsigned long nrColors;//set to 0 for 2^n
  unsigned long impColors;//set to 0
} bmpInfoHeader_t;
```

Farbpalette

Die Farbpalette wird für Bitmaps mit einer Farbtiefe von weniger als 16Bit benötigt. Es entsteht ein Array von 2^n Einträgen zu je sizeof(bmpRGBQuad_t) Elementen. Der entsprechende Wert der am Pixel x,y steht zeigt in die Farbtabelle und erhält somit den jeweiligen Farbwert.

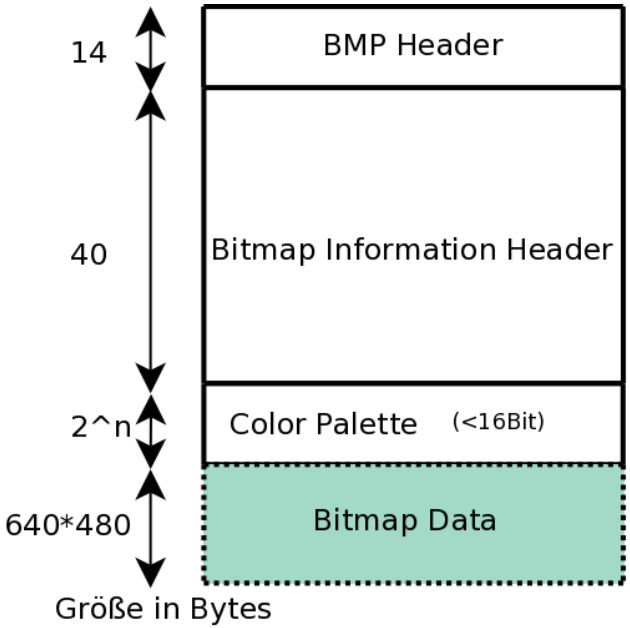


Farbpalette Definition

```
typedef struct {
   unsigned char red; //red value 0..255
   unsigned char green; //green value 0..255
   unsigned char blue; //blue vlaue 0..255
   unsigned char reserved; //not used
} bmpRGBQuad_t;
```

Bitmap Daten

Die Bitmap Daten sind Pixel für Pixel in einem Buffer abgelegt. Da wir mit einer Farbtiefe von 8Bit arbeiten entspricht jeder Array-Eintrag genau einem Pixel. Der entsprechende Wert des Array-Eintrags gibt Auskunft über die Farbe des Pixels (0x00-0xff). Die Roh-Daten können somit direkt von der Kamera als Bitmap-Daten geschrieben werden.



Beispiel zur Erzeugung von Binären Bildern

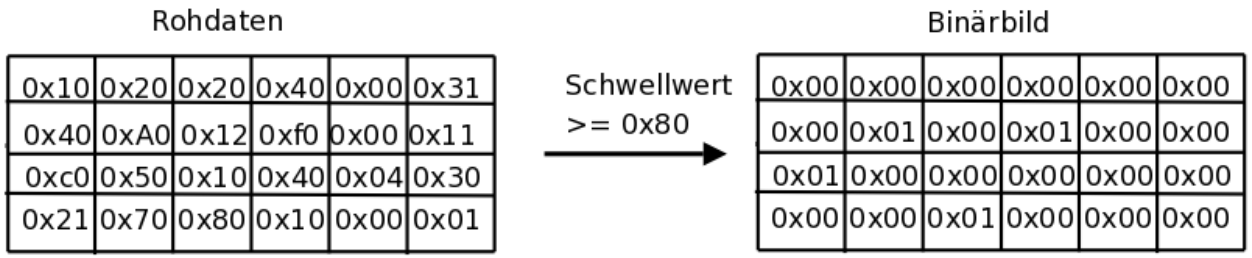
```
/* bitmap data */
static char data[640 * 480];

for (int i = 0; i < 640 * 480; i++) {
   if ((data[i] & 0xff) > LIMITVAL)
      data[i] = 0x1;
   else
      data[i] = 0x0;
}
```

Erkennung von Objekten

Nachfolgend sind 3 Schritte aufgeführt, die notwendig sind um den Ball bzw. dessen Koordinaten zu erfassen. Die einzelnen Punkte müssen in späteren Schritten in Echtzeit durchgeführt werden.

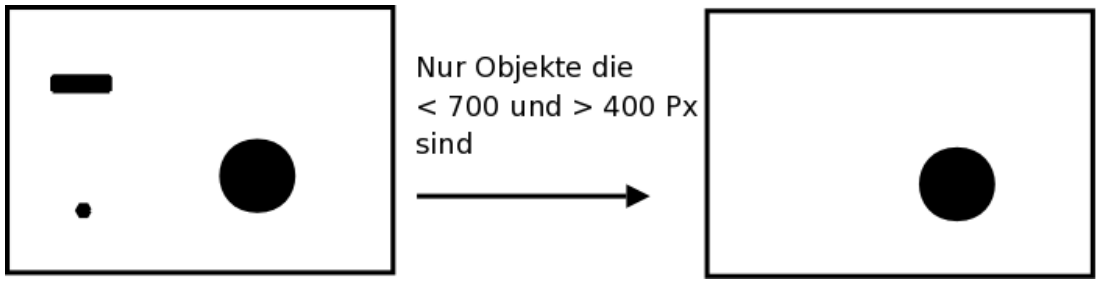
- Schwellwertanalyse: Aktive Pixel markieren (Farbwert des Pixels liegt über einem Referenzwert/Schwellwert) → Binärbild



Erkennung von Objekten

Nachfolgend sind 3 Schritte aufgeführt, die notwendig sind um den Ball bzw. dessen Koordinaten zu erfassen. Die einzelnen Punkte müssen in späteren Schritten in Echtzeit durchgeführt werden.

- Schwellwertanalyse: Aktive Pixel markieren (Farbwert des Pixels liegt über einem Referenzwert/Schwellwert) $\longrightarrow$ Binärbild
- Aneinanderliegende Pixel zu Flächen zusammenfassen und zu kleine/grosse Flächen nicht beachten.
  $minPixel \leq objekt \leq maxPixel$



Erkennung von Objekten

Nachfolgend sind 3 Schritte aufgeführt, die notwendig sind um den Ball bzw. dessen Koordinaten zu erfassen. Die einzelnen Punkte müssen in späteren Schritten in Echtzeit durchgeführt werden.

- Schwellwertanalyse: Aktive Pixel markieren (Farbwert des Pixels liegt über einem Referenzwert/Schwellwert) $\rightarrow$ Binärbild
- Aneinanderliegende Pixel zu Flächen zusammenfassen und zu kleine/grosse Flächen nicht beachten.
  $minPixel \leq objekt \leq maxPixel$
- Berechnung der X- und Y-Koordinaten des Schwerpunkts/Mittelpunkts der Fläche

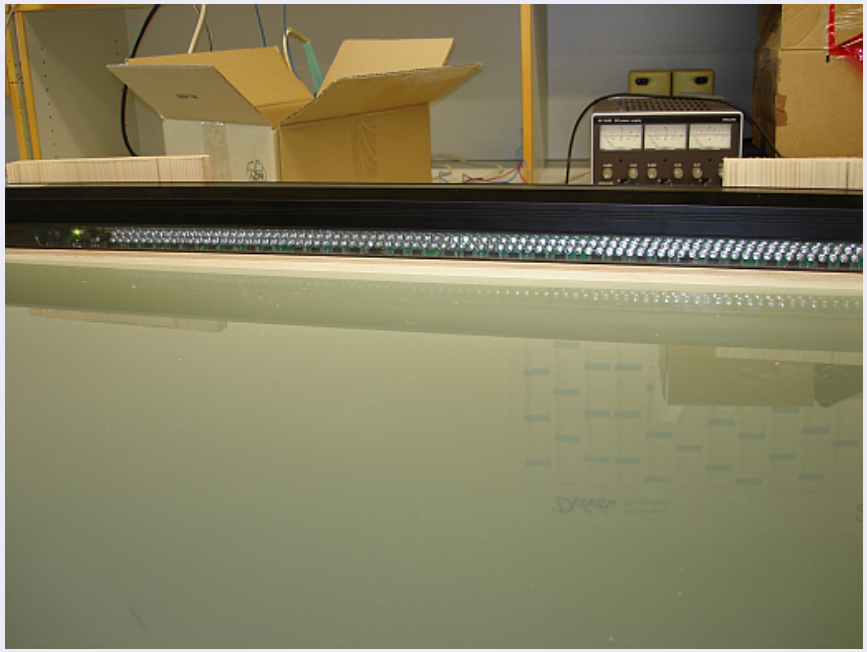
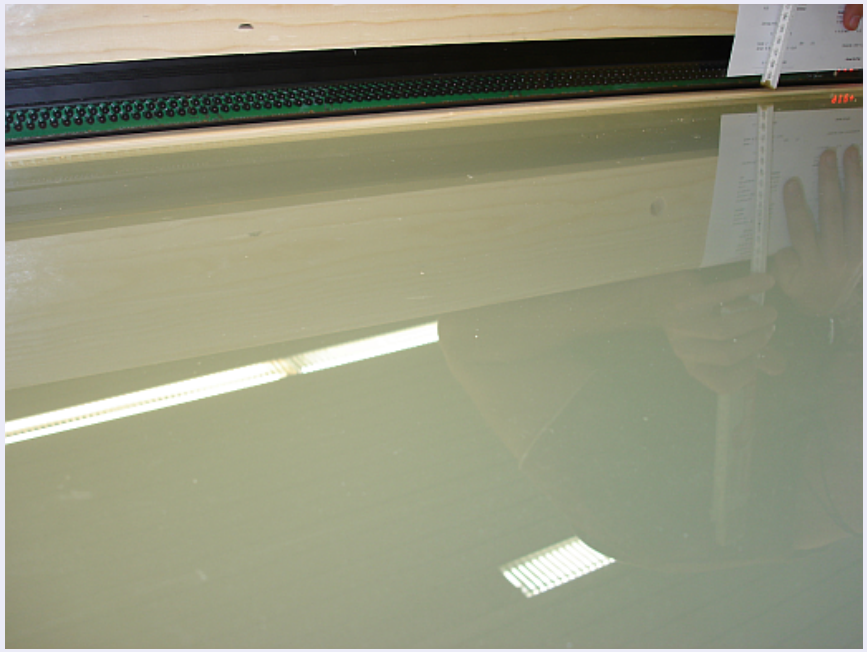
Lichtgitter

Das Lichtgitter wurde uns von der Firma Baumer für Testzwecke zur Verfügung gestellt. Die vom Lichtgitter zurückgelieferten Daten beinhalten den ersten unterbrochenen Strahl, sowie die Anzahl der unterbrochenen Strahle. Dadurch kann sich recht einfach der Mittelpunkt des Balles bestimmen lassen. Technisch ist es sehr einfach mit Sende- und Empfangsteil aufgebaut und jede LED wird nacheinander abgefragt.

- Ansteuerung via serieller Schnittstelle
  Baudrate: 9600/19200/38400, 8 Datenbits, 1 Stopbit, Parity: even
- Zykluszeit ist angegeben mit 3ms bei 38400Baud
  Herr Seck konnte mit einem Echtzeitsystem und 1000 Abfragen auf eine Antwortzeit von ca 7ms kommen. Jedoch wurde hierbei Betriebssystem bedingt nur mit 9200Baud gearbeitet.

1. Senden des Befehls (z.B. 0x81 für read_all)
2. Befehl wird zurückgegeben *echo*(lesen von 1Byte)
3. Daten werden zurückgegeben (4Byte bei read_all: Data LO/HI, Position LO/HI)

LED Sendeseite



LED Empfangsseite