

Diplomarbeit

Kameragestützte Echtzeit Objektverfolgung unter Linux.



Thomas Maurer

Version 1.0.26 html 2002

Inhaltsverzeichnis

Vorwort

1. Einleitung

2. Grundlagen

Echtzeit

1. Echtzeitbedingung - Pünktlichkeit

2. Echtzeitbedingung - Auslastung

Harte und weiche Echtzeitbedingung

Farben

Das Auge verglichen mit einer Kamera

Das Erkennen von Farben im Auge

Das RGB-Farbsystem

Das CIE-Farbsystem

3. Objekterkennung/Objektverfolgung

Aufbau einer Objekterkennung

Die Beispielapplikation

Der Datenfluss

Übersicht der Implementierten Techniken

Grabbing

Segmentierung

Verarbeitungsrichtung

Statisches Thresholding

Schwellwertbestimmung mit Hilfe von Histogrammen

Automatische Schwellwertbestimmung

Adaptives Thresholding

Andere Arten der Segmentierung

Differenzbildung

Beleuchtung

Flächenerkennung

Findung der Nachbarn

Randbehandlung

V-Problematik

Objektbewertung

Mengendichte

Kreisflächenähnlichkeit

Die Selektion

Subframing / Verfolgung

4. Grafisches-Fenster

X-Window-Client-Server-Prinzip

X-Window-Nachrichten

Xlib-Fenster

5. Diskussion und Bewertung

System-Auslastung

Methoden-Auslastung

Algorithmen

Tests

Weiterentwicklung und Verbesserung

6. Zusammenfassung und Ausblick

A. Konfiguration des Systems

B. Bedienung der Beispielapplikation

C. Quellcode der Beispielapplikation

D. Rechtliches / Eingetragene Warenzeichen

Glossar

Literatur

Stichwortverzeichnis

Abbildungsverzeichnis

1-1. Möglicher Laboraufbau

2-1. Reaktionsbereich

2-2. Formel der ersten Echtzeitbedingung

2-3. Formel der zweiten Echtzeitbedingung

2-4. Auslastung

2-5. Kostenfunktion einer harten und weichen Echtzeitbedingung

2-6. Nutzenfunktion einer harten, weichen und nicht Echtzeitbedingung

2-7. Das Auge [UniWup1998]

2-8. Die Kamera

2-9. Die Retina [ScoBew1997]

2-10. Der RGB-Farbraum

2-11. Der RGB-Farbraum und der sichtbare Farbraum [Hartl2002]

- 2-12. CIE-Normfarbtafel [Seilnacht2002]
- 3-1. Aufbau einer Objekterkennung
- 3-2. Datenflussdiagramm der Beispielapplikation
- 3-3. Verschachteltes Grabben
- 3-4. Verschachteltes Grabben Version 2
- 3-5. Die Auslastung beim verschachtelten Grabben
- 3-6. Die Auslastung beim Verschachtelten Grabben mit Deadline-Verletzung
- 3-7. Die Verarbeitungsrichtung der Segmentierung und der Flächenerkennung
- 3-8. Threshold über die Summe der Farben
- 3-9. Beispielcode: Threshold über die Summe der Farben
- 3-10. Summierender Threshold auf ein Bild angewendet
- 3-11. Zweistufiges Threshold für jede der drei Farben
- 3-12. Beispielcode: Zweistufiges Threshold für jede der drei Farben
- 3-13. Wirkungsweise eines zweistufigen Threshold auf jede der drei Farben
- 3-14. Grauwert-Histogramm des Ballonbildes
- 3-15. Bewertung von Histogrammen
- 3-16. Formel zur Errechnung eines Mittelwertes
- 3-17. Veranschaulichung des Isodata Algorithmus
- 3-18. Isodata Algorithmus zur Bestimmung eines einfachen Schwellwertes
- 3-19. Eine Kante mit ihrer ersten und zweiten Ableitung
- 3-20. Veranschaulichung eines Image und eines Kernel
- 3-21. Berechnung einer Convolution
- 3-22. Formel zur Berechnung einer Convolution
- 3-23. Typische Kernel
- 3-24. Formel zu Differenzbildung mit Offset
- 3-25. 4- und 8-connectivity
- 3-26. 4-connectivity Nachbarerkennung
- 3-27. 8-connectivity Nachbarerkennung
- 3-28. Testbild zur V-Problematik
- 3-29. Größtmögliche Vereinzelung bei der 8-connectivity
- 3-30. LABELS Tabelle
- 3-31. Objekte zur Auswahl
- 3-32. Formel zur Errechnung der Dichte
- 3-33. Formel zur Errechnung der Mengendichte
- 3-34. Änderung der Dichte bei Drehung
- 3-35. Formel der Kreisflächenähnlichkeit
- 3-36. Bearbeitungsbereiche beim Subframing
- 3-37. Platzierung des Subframe
- 3-38. Verlust des Objektes
- 4-1. drei Papierkugeln in Subframes
- 4-2. Das X-Window-Client-Server-System
- 5-1. System und Benutzer Last
- 5-2. drei Papierkugeln in Subframes
- 5-3. drei mal das gesamte Bild
- 5-4. System und Benutzer Last bei Verwendung der USB-Kamera
- 5-5. drei Papierkugeln in Subframes mit einer USB-Kamera aufgenommen
- 5-6. drei Mal das gesamte Bild mit einer USB-Kamera aufgenommen
- 5-7. Profil-Informationen (Auszug) der Beispielapplikation
- 5-8. Profil-Information (Auszug) der Beispielapplikation bei schlechter Wahl der Threshold-Level
- 5-9. Profil-Informationen (Auszug) der Beispielapplikation bei einer USB-Kamera

Diplomarbeit

- 5-10. Profil-Information (Auszug) der Beispielapplikation bei schlechter Wahl der Threshold-Level und USB-Kamera
 - 5-11. Threshold- und Labeling-Testframes
 - 5-12. Tracing-Testframes
 - 6-1. Verfolgung der Rennwagen auf der Modell-Rennbahn
 - B-1. Aufbau einer Objekterkennung
 - C-1. Quellcode der Beispielapplikation
-

Nach vorne
Vorwort

Vorwort

Betrachtet man die fünf Sinne eines Menschen, so gehört das Sehen zu den für uns bedeutendsten Wahrnehmungen. Der Vorteil des Sehens liegt in der kontaktlosen und schnellen Aufnahme einer hohen Informationsmenge. Unser Gehirn leistet bei der optischen Informationserfassung erstaunliches. Das Sehen erlaubt dem Menschen in sekundenschnelle mehrere Objekte in seiner Umgebung zu erkennen. Hierbei werden unterschiedliche Ausleuchtungsverhältnisse oder eine Veränderung des Betrachtungswinkels in weiten Grenzen problemlos durch das Auge und das Gehirn zugelassen, ohne die Objekterkennung sonderlich zu beeinflussen. Dieses System weist eine so hohe Flexibilität und Leistungsfähigkeit auf, dass es bis zum heutigen Stand der Technik noch keine vergleichbare Nachbildung gibt.

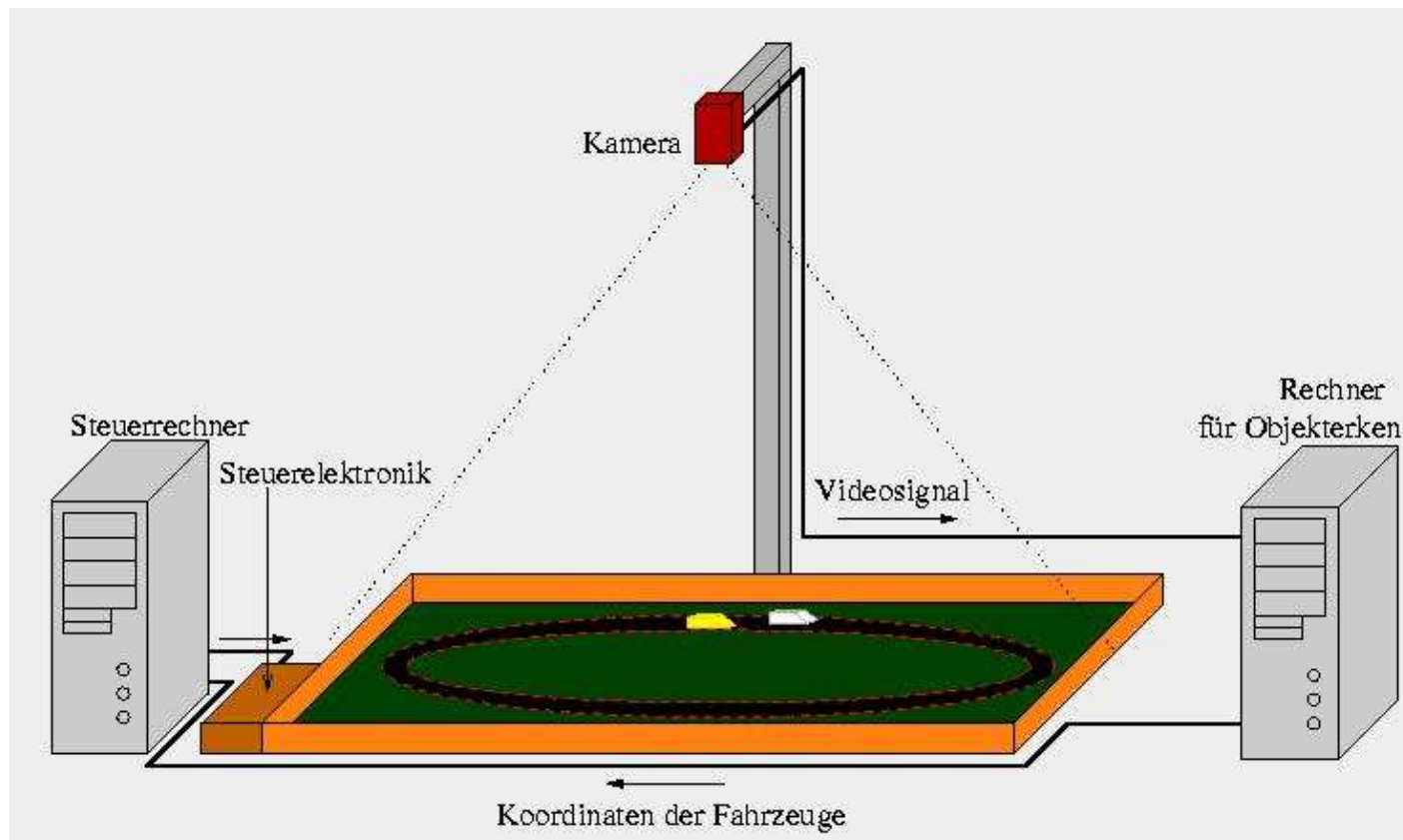
Gerade durch die hohe Informationsmenge, die bei der optischen Datenverarbeitung anfällt, besteht in diesem Forschungsgebiet ein extremes Verlangen nach einem sehr hohen Maß an Rechenleistung. Mit der Entwicklung von leistungsfähigen Computersystemen in den vergangenen Jahren hat das Thema der optischen Informationsverarbeitung immer mehr an Bedeutung gewonnen. Dazu kommt, dass auch bei der Entwicklung von Halbleitern zur Bilderfassung immer höhere Auflösungen, Bildraten und Empfindlichkeiten erreicht werden können, so dass wir heute schon in der Lage sind, mit Computersystemen die Leistung des menschlichen Sehens und Erkennens in speziellen Gebieten und unter speziellen Bedingungen zu erreichen und zu übertreffen.

Kapitel 1. Einleitung

An der Hochschule Niederrhein - Standort Krefeld, in dem Labor Echtzeit-Systeme, befindet sich eine Modell-Rennbahn, deren Fahrzeuge sich durch Computerprogramme steuern lassen. Mit Lichtschranken, die an Schlüsselpositionen in die Fahrbahn eingelassen sind, lässt sich die aktuelle Position der Rennwagen feststellen. Nebenbei kann das Schleuderverhalten der Fahrzeuge in zwei Kurven ermittelt werden. Diese Aufgabe übernimmt eine im Dach der Wagen montierte Leuchtdiode und eine Reihe von Fototransistoren über den Kurven.

Das Ziel meiner Arbeit ist es die "einfache" Realisierbarkeit einer kameragestützten Echtzeit Objektverfolgung unter Linux zu überprüfen, wie sie beispielsweise zur Beobachtung der Fahrzeuge auf der Modell-Rennbahn benötigt wird. Einfach bedeutet, ohne besonderen Hardwareaufwand. So soll ein 800 MHz PC, handelsübliche Kameras und eine unter dem Betriebssystem Linux lauffähige Software verwendet werden. Erweist sich eine solche Objektverfolgung als realisierbar, soll eine technische Grundlage geschaffen werden, mit deren Hilfe sich eine Objektverfolgung für beispielsweise die Rennbahn wie sie in Abb. Möglicher Laboraufbau dargestellt ist, umsetzen lässt.

Abbildung 1-1. Möglicher Laboraufbau



Einleitung

Der Schwerpunkt der Diplomarbeit liegt auf der Entwicklung einer beispielhaften Applikation, die die notwendigen Techniken und Grundlagen aufzeigt, welche für eine Realisierung einer Positionserkennung der Rennwagen auf ihrer Strecke benötigt wird.

Da diese Erkennung unter Echtzeitbedingungen zu erfolgen hat, wurde bei der Realisierung von Algorithmen und Programmen darauf geachtet, Rechenleistung und Verarbeitungszeit einzusparen, ohne die Lesbarkeit des Codes zu stark zu senken.

Bei der Gestaltung dieser Arbeit wurde auf eine praxisnahe und leichtverständliche Form Wert gelegt. So werden in dem Kapitel Grundlagen allgemeinen Kenntnisse vermittelt, die für diese Arbeit von Bedeutung sind. Danach gliedert sich in dem Kapitel Objekterkennung/Objektverfolgung die Arbeit in die verschiedenen Abläufe zur Erkennung und Verfolgung eines Objektes. An diesen Stellen wird verstärkt auf den Quellcode verwiesen, um eine Umsetzungsmöglichkeit aufzuzeigen oder auf sie einzugehen.

Beendet wird die Arbeit durch die Kapitel Diskussion und Bewertung und Zusammenfassung und Ausblick. In diesen wird unter anderem vorgestellt, wie sich die Verarbeitungszeiten in der Beispielapplikation und im Rechensystem verteilen, wie die Applikation getestet wurde, wofür die Beispielapplikation, oder eine Weiterentwicklung dieser, eingesetzt werden kann, sowie wo ihre Stärken und Schwächen liegen.

Darüber hinaus verfügt diese Arbeit über ein Stichwortverzeichnis und ein umfangreiches Glossar, in den Fachbegriffe und kurze Erklärungen nachgeschlagen werden können.

Zurück
Vorwort

Zum Anfang

Nach vorne
Grundlagen

Kapitel 2. Grundlagen

Inhaltsverzeichnis

[Echtzeit](#)[Farben](#)

Echtzeit

Untersucht man eine Echtzeit Objektverfolgung ergeben sich zwei Anforderungen, die erfüllt werden sollten.

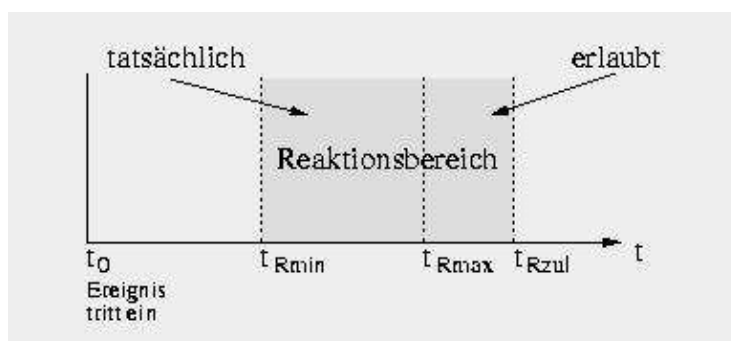
1. Es soll gewährleistet werden, dass eine gewisse Anzahl von Bildern pro Sekunde verarbeitet werden. Hierbei ist wichtig, dass zwischen zwei Bildern nur eine gewisse Zeit zur Verfügung steht, in der ein Bild bearbeitet werden kann. Danach muss mit der Bearbeitung des nächsten Bildes begonnen werden, und die Bearbeitung des vorhergehenden Bildes abgeschlossen sein.
2. Zum anderen muss das System so viel Rechenleistung zur Verfügung stellen können, dass es alle ihm gestellten Aufgaben bearbeiten kann.

In diesen Punkten lassen sich die beiden Echtzeitbedingungen erkennen.

1. Echtzeitbedingung - Pünktlichkeit

Wichtig ist, dass eine Aufgabe innerhalb einer gewissen Zeitspanne fertig wird.

Abbildung 2-1. Reaktionsbereich



Pünktlichkeit bedeutet hier,

1. dass die Aufgabe nicht vor einem bestimmten Zeitpunkt t_{Rmin} erledigt wurde.
2. dass die Aufgabe bis zu einem bestimmten Zeitpunkt t_{Rzul} erledigt wird.

In der Abb. Reaktionsbereich ist noch eine dritte Zeit, die maximale Reaktionszeit t_{Rmax} , zu sehen. Sie muss immer kleiner oder gleich der maximal zulässigen Reaktionszeit t_{Rzul} sein. Die maximale Reaktionszeit ist die relative Zeit, gemessen ab Eintritt des Ereignisses t_0 , die der Prozess im ungünstigsten Fall (worst case) benötigt, um auf das Ereignis zu antworten.

Soll ein Ereignis die erste Echtzeitbedingung erfüllen, so muss die Reaktionszeit t_R zwischen t_{Rmin} und t_{Rzul} liegen. Siehe hierfür Abb. Formel der ersten Echtzeitbedingung.

Abbildung 2-2. Formel der ersten Echtzeitbedingung

$$t_{Rmin} \leq t_R \leq t_{Rzul}$$

2. Echtzeitbedingung - Auslastung

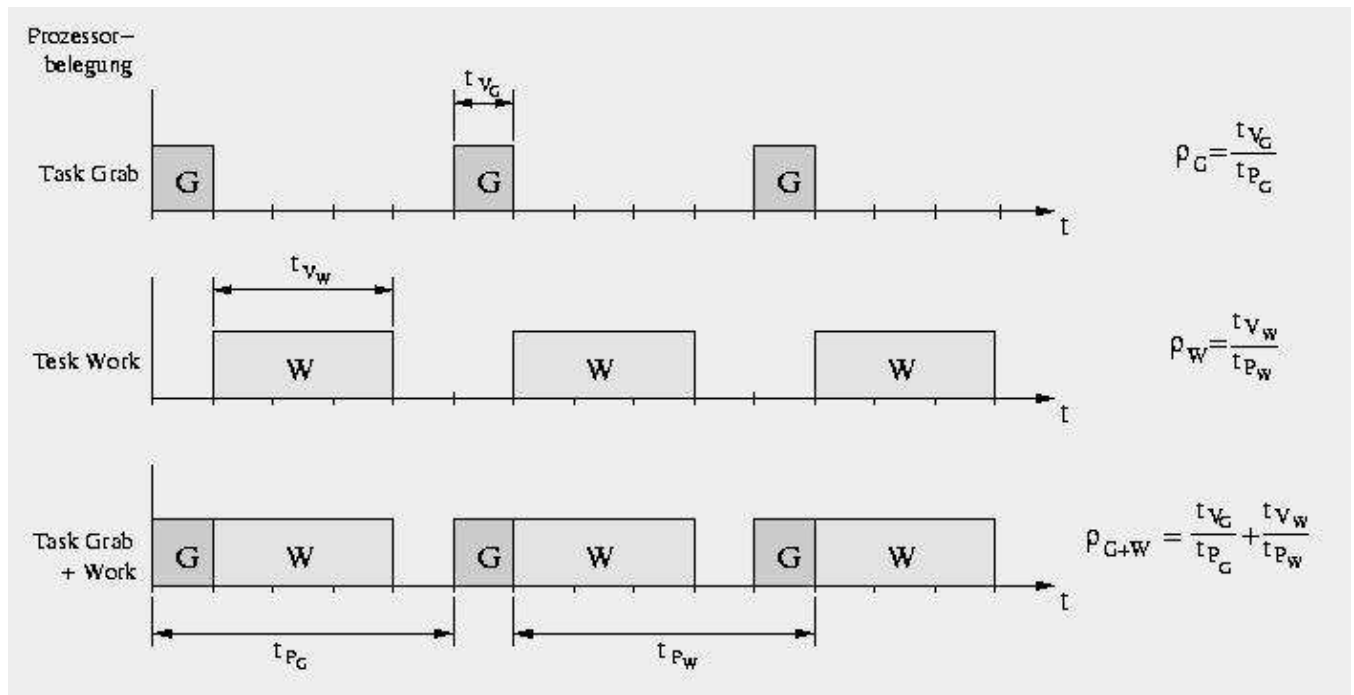
Ein weiterer Punkt, den ein Echtzeitsystem erfüllen muss, ist die Summe aller Aufgaben bearbeiten zu können. Dazu gehört auch die Bearbeitung paralleler Prozesse. Das bedeutet, dass die Gesamtauslastung des Systems kleiner oder gleich 100% sein muss. Dies ist in der Abb. Formel der zweiten Echtzeitbedingung zu sehen. Hierbei ist mit t_v die Verarbeitungszeit gemeint, also die Zeit, die der Prozess die CPU benutzt. Dagegen wird mit t_p die Prozesszeit bezeichnet. Sie ist die Zeit, die der Prozess hat, bis er abgearbeitet sein muss.

Abbildung 2-3. Formel der zweiten Echtzeitbedingung

$$\rho = \sum_{i=0}^n \frac{t_{v_i}}{t_{p_i}} \leq 1$$

In Abbildung Auslastung wird die Auslastung eines Echtzeitrechners angezeigt, auf dem die Task "Grab" und die Task "Work" laufen. Zuerst grabt die Task-Grab ein Bild von einem Input Device, welches danach von der Task-Work nach Objekten durchsucht wird. Dieser Vorgang wird fortlaufend wiederholt. Wichtig ist hierbei, dass die Gesamtauslastung ρ_{G+W} kleiner oder gleich 1 bleibt.

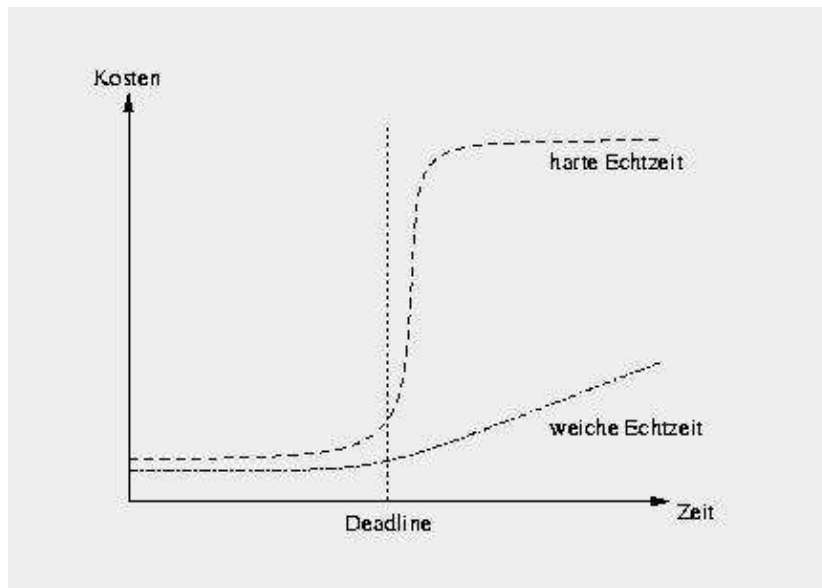
Abbildung 2-4. Auslastung



Harte und weiche Echtzeitbedingung

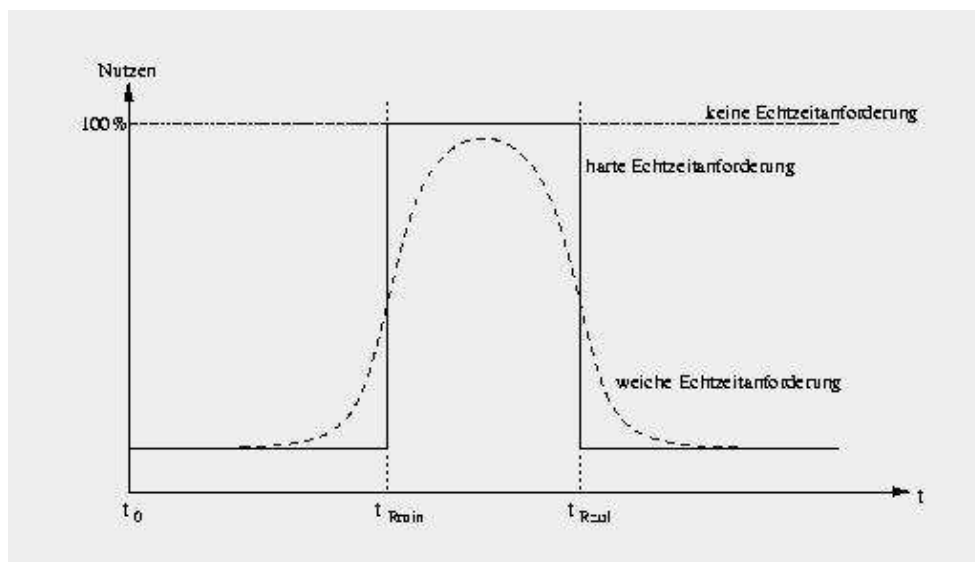
Nicht bei jedem System ist die Anforderung nach Rechtzeitigkeit gleich hoch. Während es bei einem Objekterkennungssystem, das eine Spielzeugrennbahn überwacht, relativ ungefährlich ist, wenn ein Bild nicht rechtzeitig bearbeitet und dadurch verworfen wird, ist es sehr viel schlimmer und ggf. mit hohen Personen und Sachschäden verbunden, wenn bei einem Andockmanöver an die ISS eine Steuerröhre des Spaceshuttle zu spät abgeschaltet wird. Führt das Überschreiten einer Deadline zu Personen oder Sachschäden, spricht man von einer *harten Echtzeitbedingung*. Sind die Folgen hinnehmbar, spricht man von einer *weichen Echtzeitbedingung* (siehe Bild Kostenfunktion einer harten und weichen Echtzeitbedingung).

Abbildung 2-5. Kostenfunktion einer harten und weichen Echtzeitbedingung



In dieser Abbildung sieht man, dass bei einer *weichen Echtzeitbedingung* das Überschreiten der Deadline zu einem leichten Kostenanstieg führt, während die Kosten sich bei der *harten Echtzeitbedingung* nach Verletzung der Deadline sprunghaft erhöhen. Eine andere Darstellung ist mit Hilfe der Nutzenfunktion (Abb. Nutzenfunktion einer harten, weichen und nicht Echtzeitbedingung) möglich.

Abbildung 2-6. Nutzenfunktion einer harten, weichen und nicht Echtzeitbedingung



Bei der Nutzenfunktion ist zu erkennen, dass bei der *harten Echtzeitbedingung* die Reaktion in dem Zeitraum zwischen t_{Rmin} und t_{Rzul} erfolgen muss um einen Nutzen zu erzielen. Bei der *weichen Echtzeitbedingung* sinkt der Nutzen je weiter er von der optimalen Reaktionszeit abweicht. Die Grenzen müssen hier nicht so genau eingehalten werden, um einen Restnutzen zu behalten.

Da es zwischen harter und weicher Echtzeitbedingung eine Vielzahl von Abstufungen gibt, die von der jeweiligen Aufgabenstellung abhängig sind, ist nicht immer eine eindeutige Einordnung möglich. Für weitere

Grundlagen

Informationen sei hier auf die Quelle [Quade2001] verwiesen.

[Zurück](#)

Einleitung

[Zum Anfang](#)

[Nach vorne](#)

Farben

Farben

Gerade für die Entwicklung einer Objekterkennung / -verfolgung, bei der Farbbilder von einer Kamera aufgenommen, Farbfilter auf das zu untersuchende Objekt angewendet oder bei der ein Bild in einem gewissen Farbformat im Speicher ablegt wird, ist es wichtig, ein Verständnis für Farben zu entwickeln.

Die Farbe, in der ein Mensch ein Objekt wahrnimmt, hängt nicht allein von dem betrachteten Objekt ab, sondern ebenfalls von der Lichtquelle, die das Objekt beleuchtet, von der Farbe des Umfeldes und der Beschaffenheit und Farbe des Mediums, in dem sich das Objekt befindet (z.B. Luft). Ein besonders schwer zu bemessender Einfluss auf das Wahrnehmen von Farben geht vom menschlichen Auge und Gehirn aus. So ist nicht nur nachgewiesen, dass Farben subjektiv von verschiedenen Menschen unterschiedlich wahrgenommen werden, sondern dass sich die Farbwahrnehmung im Laufe des Lebens eines Menschen verändert.

Das Auge verglichen mit einer Kamera

Betrachtet man den Vorgang der Bildaufnahme bei einer Videokamera oder beim menschlichen Auge, fällt auf, dass beide nach einem ähnlichen Prinzip arbeiten. Bei beiden Verfahren wird das Bild mittels einer Linse verkleinert und auf beiden Achsen umgekehrt auf ein Feld von Fotosensoren projiziert. Von dort wird es zur Weiterverarbeitung beim Sehen an das Gehirn oder bei der Aufnahme durch eine Kamera an einen Rechner geleitet.

Abbildung 2-7. Das Auge [UniWup1998]

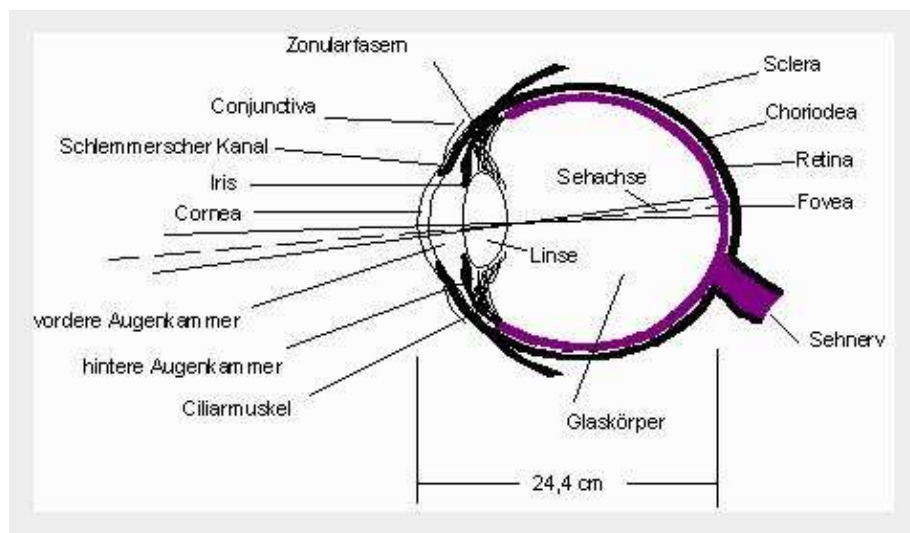
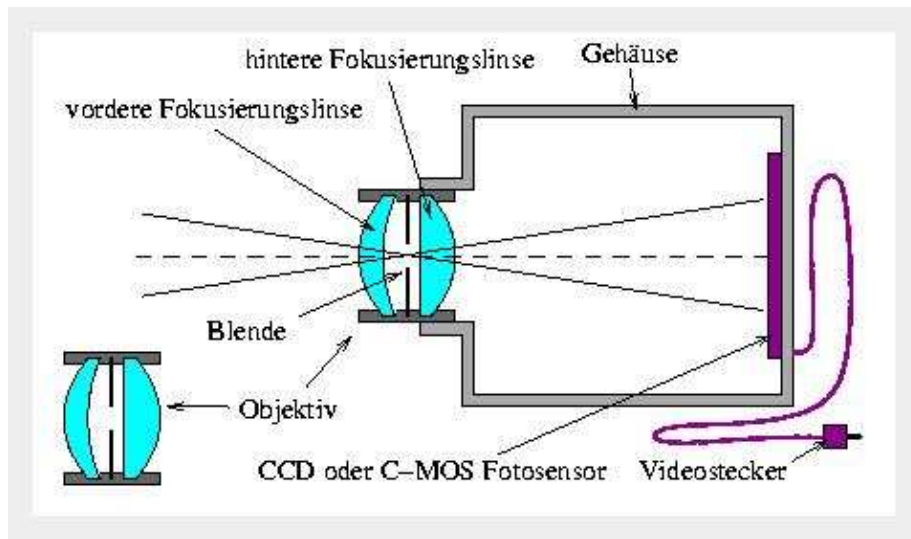


Abbildung 2-8. Die Kamera



Vergleicht man den Aufbau des Auges (Abb. *Das Auge [UniWup1998]*) mit dem Aufbau einer Kamera (Abb. *Die Kamera*) genauer, erkennt man schnell, dass der Mensch bei der Entwicklung der Kamera das Auge als Vorbild verwendet hat. In der Folgenden Auflistung sind beide Systeme einander gegenübergestellt.

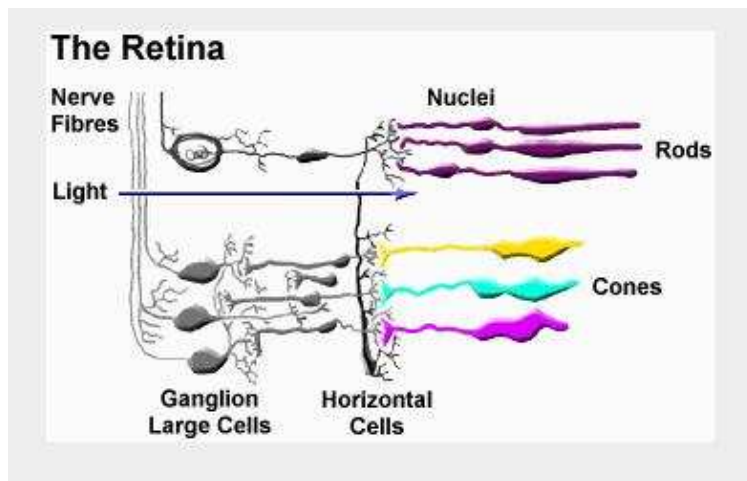
- Die grobe Fokussierung des Bildes auf die Netzhaut erfolgt bei dem Auge über die *Cornea* und die *vordere Augenkammer*, bei der Kamera übernimmt diese Aufgabe die *vordere Fokussierungslinse*.
- Die Aufgabe der feinen Fokussierung übernimmt im Auge die *Linse*. Sie ist an *Zonularfasern* befestigt, die wiederum zu dem *Ciliarmuskel* führen. Bei der Kontraktion dieses Ringmuskels nimmt die *Linse* eine gekrümmte Form an. Wird der Ringmuskel entspannt, sorgt die Spannung der *Zonularfasern* und der Innendruck des Auges dafür, dass die Linse eine abgeflachte Form annimmt. Durch diese Veränderung der Linsenform kann das durch sie fallende Licht unterschiedlich stark gebrochen werden. Bei der Kamera ist die Veränderung der Linsenform nur schwer zu realisieren, deshalb wird hier zum Fokussieren des Bildes der Abstand zwischen dem *Objektiv* und dem *Fotosensor* verändert. Bei komplexeren Objektiven erfolgt eine Veränderung von Abständen zwischen mehreren Linsen.
- Um das einfallende Licht zu beschränken und bei guten Lichtverhältnissen die Tiefenschärfe zu erhöhen, verfügt das Auge über die *Iris*. Durch das Zusammenziehen der *Iris* verkleinert sich die runde Öffnung, durch die Licht in das Augeninnere gelangt. Das gleiche Prinzip wird bei einer Kamera durch die *Blende* umgesetzt.
- Die Umwandlung von Lichtsignalen in elektrische Nervenimpulse übernimmt beim Auge die *Retina* (Netzhaut). Sie kleidet die komplette Rückwand des Auges aus und ist an der Stelle gegenüber der *Linse*, dieser Bereich wird *Vovea* genannt, besonders dicht mit Fotosensoren besetzt. Bei einer Kamera befindet sich eine Matrix aus *CCD oder C-MOS Sensor* an der dem *Objektiv* gegenüberliegenden Gehäusewand, die ebenfalls das einfallende Licht in elektrische Signale wandelt.
- Für den Weitertransport der Informationen werden beim Auge die Nervenimpulse über den *Sehnerv* an das Gehirn weitergeleitet. Eine ähnliche Aufgabe hat der *Videoausgang* der Kamera, der das Bild zur weiteren Verarbeitung an einen Computer leitet.
- Für die mechanische Stabilität des Apparates sorgt beim Auge die *Sclera* (Lederhaut) und der *Glaskörper*. Bei einer Kamera wird diese Aufgabe von dem *Gehäuse* übernommen.

(Vergleiche [UniWup1998] und [Möbes1999]).

Das Erkennen von Farben im Auge

Da die Farberkennung in speziellen Zellen der *Retina* (Netzhaut) stattfindet, ist es interessant, diese Zellen genauer zu betrachten (Abb. *Die Retina [ScoBew1997]*). Es fällt auf, dass die *Retina* verkehrtherum aufgebaut ist. So muss das Licht, um an die Sensorzellen zu gelangen, erst ein Netz aus Nervenleitungen durchdringen. Nur am Fovea (Punkt gegenüber der Linse) liegen die Sensorzellen frei.

Abbildung 2-9. Die Retina [ScoBew1997]



Die *Retina* verfügt über zwei Arten von Sensorzellen. Die *rods* (Stäbchen), welche sehr empfindlich gegenüber geringen Lichtmengen sind, dafür aber nur bis zu einer gewissen Lichtmenge eine Zunahme der Helligkeit feststellen können. Die *rods* können nur zwischen Helligkeitsstufen unterscheiden. Sie sind nicht in der Lage Farbunterschiede wahrzunehmen. Auf der *Retina* sind diese 120 bis 150 Millionen Sensoren bevorzugt im peripheren Bereich verteilt. Im Bereich der *Fovea* befindet sich verstärkt die zweite Sorte von Sensorzellen, die *cones* (Zäpfchen). Sie sind weniger empfindlich und eignen sich daher für höhere Lichtstärken. Die 6 bis 7 Millionen *cones* sind in drei Gruppen unterteilt, wovon jede für eine bestimmte Wellenlänge beziehungsweise eine der drei Farben, Violett, Grün und Gelb besonders empfindlich ist. Nur durch eine aufwendige Umrechnung in menschlichen Gehirn, ist der Mensch in der Lage, die Farben Rot, Grün und Blau wahrzunehmen.

Schon in der *Retina* beginnt der menschliche Seh-Apparat mit einer Auswertung der Bildinformation. Die zur optischen Achse querverlaufenden *horizontal cells* (horizontalen Zellen) führen eine Differenzbildung zwischen den von den Sensorzellen kommenden Signalen durch. So erfolgt schon im Auge eine Kantenerkennung. Die *ganglion cells* (Ganglion Zellen) sind es schliesslich, die die Nervenimpulse weiter zum Sehnerv und somit zum Gehirn leiten.

Zum Vergleich siehe [ScoBew1997] und [Möbes1999].

Das RGB-Farbsystem

Farben

Bei der Vermischung von Farben gibt es zwei verschiedene Techniken, zum einen die *Subtraktive Farbmischung*, bei der sich aus den drei Farben Blau, Rot und Gelb fast (siehe Kapitel Das CIE-Farbsystem) alle anderen Farben erzeugen lassen. Diese Art von Farbmischung tritt auf, wenn sich verschiedenfarbige Stoffe vermischen oder überlagern. Je mehr Farbanteile der drei Additiven Grundfarben an der Bildung der Farbe beteiligt sind, um so dunkler wird das Ergebnis.

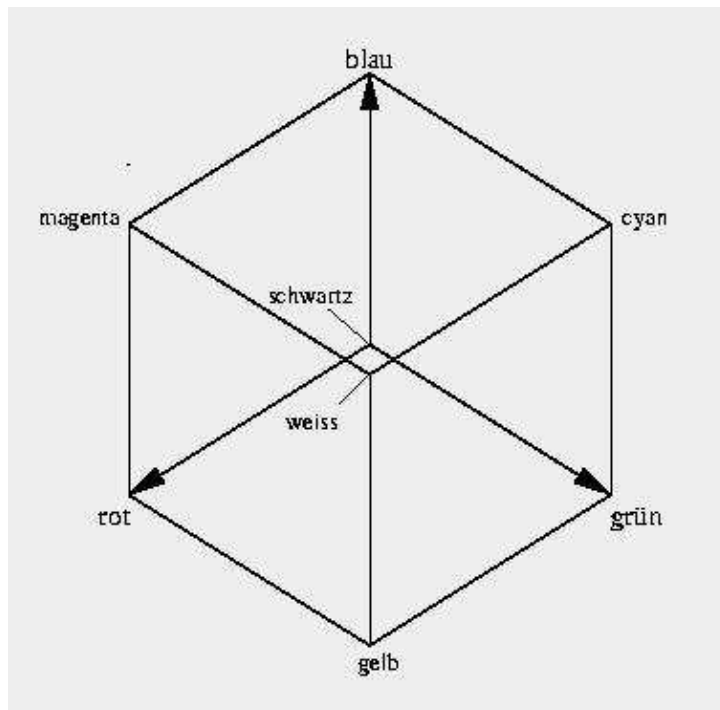
Die zweite Technik ist die *Additive Farbmischung*. Bei dieser wird das Licht von drei Lichtquellen mit den Farben Rot, Grün und Blau überlagert. Werden diese drei Farben auf eine weisse Fläche projiziert, vermischen sie sich zu Weiss. Durch Variation der Lichtanteile lassen sich so fast (siehe Kapitel Das CIE-Farbsystem) alle sichtbaren Farben erzeugen.

Das RGB-Format beschreibt eine Art, die die Farb- und Helligkeitsinformation eines Bildpunktes nach dem Prinzip der *Additiven Farbmischung* ablegt. Das bedeutet, dass jeder Punkt eines Bildes durch die Anteile der Farben Rot, Grün und Blau beschrieben wird. Ein Punkt, bei dem alle drei Anteile null sind wird Schwarz dargestellt.

Eine Möglichkeit fast (siehe Kapitel Das CIE-Farbsystem) jede mögliche Farbe in einem dreidimensionalen Raum darzustellen, stellt der der RGB-Farbraum dar (siehe Abb. Der RGB-Farbraum).

Im RGB-Farbraum kann jede dieser Farben durch ihre Rot-, Grün- und Blauanteile dargestellt werden. Eine andere Technik verwendet einen Vektor. Die Intensität der einzelnen Farben wird durch die Länge des Vektors angegeben. Die Farbe ergibt sich durch die Winkel, welche die Ausrichtung des Vektors bestimmen.

Abbildung 2-10. Der RGB-Farbraum

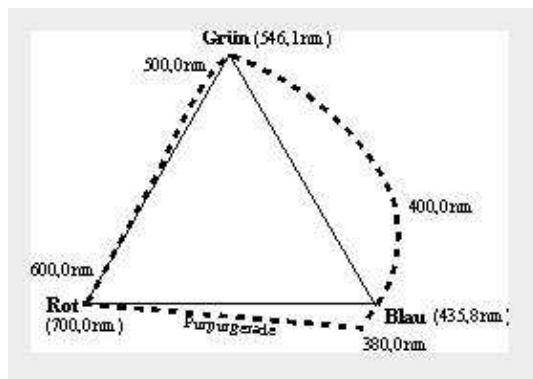


Das CIE-Farbsystem

Wie im vorhergehenden Text schon erwähnt, lässt sich mit einem Farbsystem, das auf drei reellen Grundfarben (Farben wie sie in der Natur vorkommen) basiert, nicht jede Farbe erzeugen. Praktisch lässt sich das durch einen Versuch mit vier Lichtquellen überprüfen. Eine der Lichtquellen, die Referenzlichtquelle, leuchtet in einer bestimmten Farbe auf eine weiße Fläche. Mit den drei anderen Lichtquellen, die die Grundfarben Rot, Grün und Blau haben, sollte sich jetzt die Farbe der Referenzlichtquelle darstellen lassen. Für einige Farben lässt sich dies nur erreichen, wenn die Referenzlichtquelle mit einer der drei Grundfarben gemischt wird. Anders ausgedrückt: Eine der drei Grundfarben müsste zur Darstellung der Farbe mit einem negativen Wert in die RGB-Mischung einfließen.

In der folgenden Abbildung (Abb. Der RGB-Farbraum und der sichtbare Farbraum [Hartl2002]) ist das RGB-Farbsystem in Dreiecks-Koordinaten abgebildet. Jede der Ecken repräsentiert eine Grundfarbe. Im Inneren des Dreieckes befinden sich alle Farben, die sich durch eine Mischung der drei Grundfarben darstellen lassen. Die gestrichelte Linie, die das Dreieck umgibt, stellt den Farbraum dar, den das menschliche Auge wahrnehmen kann. Man erkennt, dass mit dem RGB-Farbsystem nur ein gewisser Anteil des Farbspektrums darstellbar ist.

Abbildung 2-11. Der RGB-Farbraum und der sichtbare Farbraum [Hartl2002]

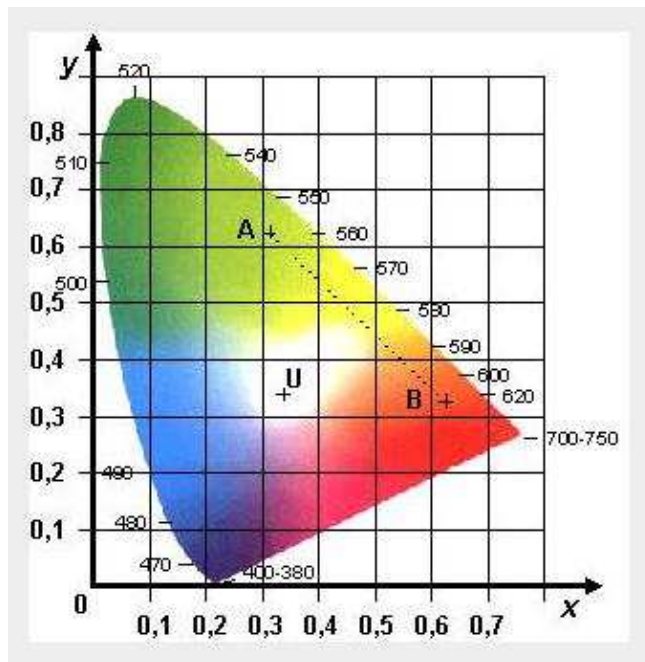


Um ein System zu erhalten, in dem sich das komplette Spektrum befindet, wurden spezielle virtuelle Farben entwickelt, die sogenannten *virtuellen Primärvalenzen*. Diese drei Farben, mit der Bezeichnung X, Y und Z, spannen ein gleichseitiges dreieckiges Koordinatensystem auf, in dem sich alle sichtbaren Farben befinden. Versucht man diese *virtuelle Primärvalenzen* zu benennen oder in RGB-Anteilen auszudrücken, ergibt sich folgende Bezeichnung:

- X ist ein super-gesättigtes Purpurrot mit den RGB-Anteilen: $+ 2,36460 \text{ Rot} - 0,51515 \text{ Grün} + 0,00520 \text{ Blau}$
- Y ist ein super-gesättigtes Grün mit den RGB-Anteilen: $- 0,89653 \text{ Rot} + 1,42640 \text{ Grün} - 0,01441 \text{ Blau}$
- Z ist ein super-gesättigtes Blau mit den RGB-Anteilen: $- 0,46807 \text{ Rot} + 0,08875 \text{ Grün} + 1,00921 \text{ Blau}$

Zur einfacheren Handhabung wurde das Dreieckskoordinatensystem in ein rechtwinkliges System transformiert. Betrachtet man in diesem Raum die Fläche an der $X + Y + Z = 1$ ist, also alle Farben die gleiche Helligkeit haben, so erhält man die sogenannte *CIE-Normfarbtafel* (siehe Abb. CIE-Normfarbtafel [Seilnacht2002]).

Abbildung 2-12. CIE-Normfarbtafel [Seilnacht2002]



In diesem Farbraum stellt die äussere gebogene Linie die Wellenlänge dar und wird als *Spektralfarbentzug* bezeichnet. An ihrem Rand finden sich die Spektralfarben wieder. Für einige Farben wird auf dieser Linie die Wellenlänge angegeben. Die gerade Linie unten, die die beiden Enden verbindet, nennt man wegen ihres Rot-Blau-Übergangs *Purpurgerade*. Der Punkt U an der Koordinaten $X=Y=0,333$ des Diagramms trägt den Namen *Mittelpunktvalenz* oder *Unbuntpunkt*, da sich hier alle Farben zu Weiss ergänzen. Auf den Geraden zwischen der *Mittelpunktvalenz* und den Farben auf dem *Spektralfarbentzug* bleibt ein Farbton immer gleich. Lediglich die Sättigung der Farbe ändert sich. Je weiter sich die Farbe von der *Mittelpunktvalenz* entfernt, desto höher wird ihre Sättigung (Farbigkeit).

Zieht man eine Gerade von der Farbart A zur Farbart B (siehe Abb. *CIE-Normfarbtafel [Seilnacht2002]*), lassen sich nur die Farbarten durch *Additive Farbmischung* erzeugen, welche auf der Geraden zwischen den Punkten liegen. Man erkennt so beispielsweise, dass aus den Farben Rot und Blau nicht die Farbe Grün gemischt werden kann.

Da z.B. ein Monitor aufgrund seiner Additiven-RGB-Farbmischung nicht jede Farbe darstellen kann, ist es verbreitet, innerhalb der CIE-Normfarbentafel den Farb-Bereich, den ein optisches Ausgabegerät darstellen kann, als Fläche zu markieren.

Es soll hier angemerkt werden, dass die Farben in der Abb. *CIE-Normfarbtafel [Seilnacht2002]* nicht den echten CIE-Normfarben entsprechen können, sondern dass es sich hier nur um eine Andeutung dieser handeln kann.

Für weiterführende Informationen und zum Vergleich sei auf die Quellen [Seilnacht2002], [Möbes1999] und [Hartl2002] verwiesen.

[Zurück](#)
Grundlagen

[Zum Anfang](#)
[Nach oben](#)

[Nach vorne](#)
Objekterkennung/Objektverfolgung

Kapitel 3. Objekterkennung/Objektverfolgung

Inhaltsverzeichnis

[Aufbau einer Objekterkennung](#)

[Die Beispielapplikation](#)

[Grabbing](#)

[Segmentierung](#)

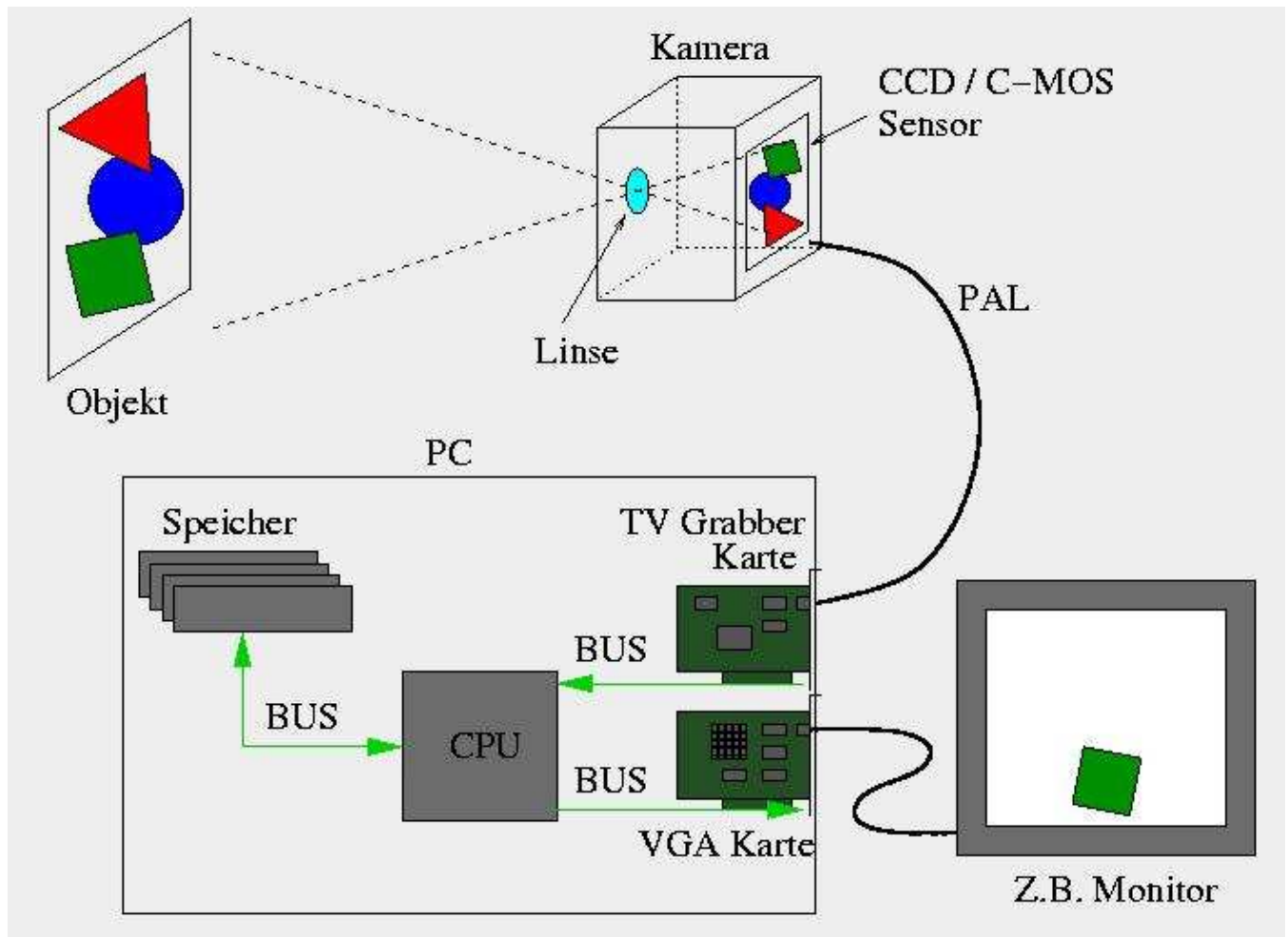
[Flächenerkennung](#)

[Objektbewertung](#)

[Subframing / Verfolgung](#)

Aufbau einer Objekterkennung

Abbildung 3-1. Aufbau einer Objekterkennung



In Abbildung Aufbau einer Objekterkennung wird veranschaulicht, wie ein Objekt mit einer Kamera aufgenommen, mit einem PC bearbeitet und auf einem Monitor ausgegeben wird. Zuerst wird das Bild des Objektes durch eine Optik (Linse) auf die Innenseite der Rückwand einer Kamera projiziert. An dieser Wand befindet sich in der Regel ein CCD oder C-MOS Sensor, der das Bild in eine elektrische Form wandelt.

Wie diese Information an den PC weitergegeben wird, hängt von der Art der Kamera ab. In dem in der Abbildung beschriebenen Fall wandelt die Kamera das Bild in ein PAL Format um, das über ein S-Video Kabel an die Frame-Grabber-Karte übertragen wird.

Eine andere weit verbreitete Technik, ist die Verwendung einer USB-Kamera. In einem solchen Fall (nicht in der Abb. Aufbau einer Objekterkennung dargestellt) würde die Bildinformationen in ein für das USB-Protokoll geeignetes Format umgewandelt, und über ein USB-Kabel an den USB-Controller des PCs gesendet werden.

Durch die API Video4Linux wird es jetzt ermöglicht, mittels System-Calls aus dem Bildstrom, den die Frame-Grabber-Karte bekommt, einzelne Bilder in einen bestimmten Speicherbereich zu kopieren. Dieser Kopiervorgang erfolgt entweder durch die CPU oder über DMA Transfer, wodurch die CPU entlastet wird.

Befinden sich die Daten im Hauptspeicher, kann die Objekterkennungssoftware damit beginnen, das gesuchte Objekt zu erkennen. Gerade diese Phase ist sehr rechenzeitaufwendig, da eine Vielzahl von Berechnungen und Vergleichen auf einen großen Speicherbereich angewendet werden. Die Größe des Speicherbereichs ergibt sich aus der Pixelzahl des eingefangenen Bildes. In Kapitel Subframing / Verfolgung wird eine Technik

gezeigt um diesen Rechenaufwand zu minimieren.

Ist die Objekterkennung abgeschlossen, wird ihr Ergebnis auf ein Ausgabedevise ausgegeben. Das kann beispielsweise bedeuten, dass ein Bild des erkannten Objekts über die Grafikkarte auf einem Bildschirm ausgegeben wird (wie in der Abbildung Aufbau einer Objekterkennung veranschaulicht). Es wäre aber auch genauso gut denkbar, dass die Koordinaten des erkannten Objekts mit Hilfe einer Netzwerkkarte an einen anderen Computer, zur weiteren Bearbeitung, übertragen werden. Da es sich bei der in dieser Diplomarbeit behandelten Objekterkennung um einen fortlaufenden Prozess handelt, wird der gesamte Ablauf ständig wiederholt. Zum Teil ist es auch möglich, Aufgaben parallel zu bearbeiten. So kann beispielsweise das nächste Bild durch DMA in den Speicher übertragen werden, während die CPU das letzte Bild nach Objekten durchsucht.

Zurück
Farben

Zum Anfang

Nach vorne
Die Beispielapplikation

Die Beispielapplikation

Bevor in den folgenden Kapiteln näher auf die verschiedenen Schritte, die für die Objektverfolgung notwendig sind, eingegangen wird, soll hier der Aufbau der Beispielapplikation erläutert werden.

Es sei hier angemerkt, dass der Begriff "Objekt" sich auf die Objekte, die die Objektverfolgung erkennt und verfolgt, bezieht. Bei "Objekte" im Sinne der Objektorientierten-Programmierung wird der Begriff "Instanz" verwendet.

Bei der Beispielapplikation handelt es sich um ein Programm, das in drei Klassen gekapselt ist. Jede dieser Klassen lässt sich einem der drei Teilgebiete des EVA-Prinzip (Eingabe Verarbeitung Ausgabe) zuordnen.

Die Klasse `video_in` übernimmt die Aufgabe der Daten-Eingabe. Sie stellt der verarbeitenden Klasse `o_tracing` ein RGB-Bild zur Verfügung. In der Klasse `o_tracing` erfolgt die Verarbeitung dieser Information in Form einer Objekterkennung/Objektverfolgung. Bevor das Bild an die Ausgabe weitergegeben wird, werden die erkannten Objekte in diesem Bild durch Eckmarkierungen hervorgehoben. Die Ausgabe des Bildes erfolgt schließlich in der Klasse `video_out`. Hier wird das Bild in einem Grafischen-Fenster auf dem Monitor ausgegeben.

Die Erzeugung der zu den Klassen gehörenden Instanzen erfolgt in der Methode `main`. Von der Eingabe-Klasse `video_in` und der Ausgabe-Klasse `video_out` wird je eine Instanz angelegt. Von der Verarbeitungs-Klasse `o_tracing` hingegen können mehrere Instanzen erzeugt werden. So ist es möglich mehr als nur ein Objekt gleichzeitig zu verfolgen. Jedem der zu verfolgenden Objekte wird eine Instanz der Klasse `o_tracing` zugeordnet.

Neben der Objekterzeugung startet die Methode `main` die Darstellung eines einfachen textorientierten Menüs, mit dessen Hilfe sich drei Instanzen der Objekterkennung/Objektverfolgung steuern lassen. Im Anhang wird auf dieses Menü *Bedienung der Beispielapplikation* weiter eingegangen.

Der Datenfluss

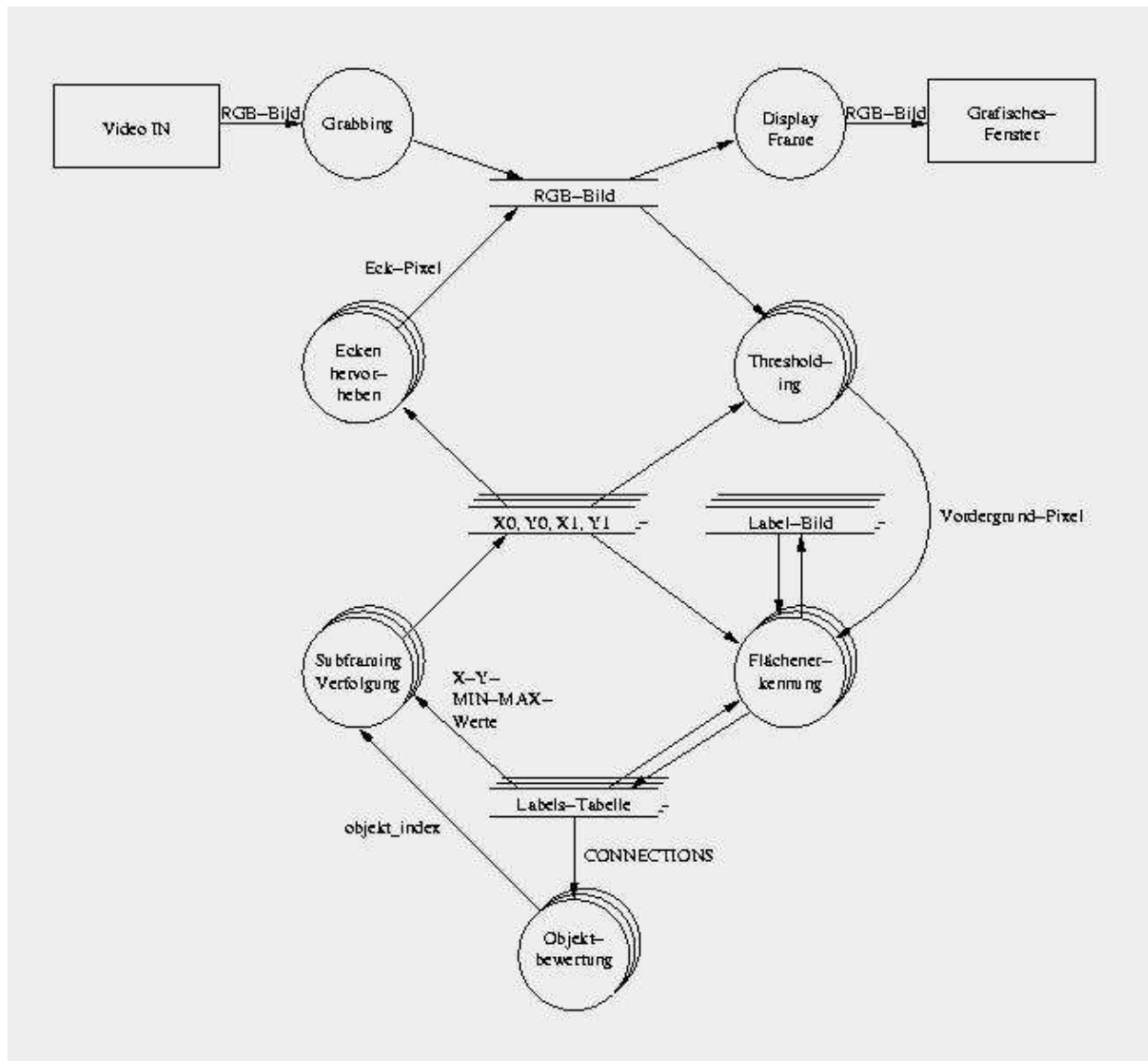
Um eine Übersicht der Datenflüsse innerhalb der Beispielapplikation zu erhalten, wird die Klasse `o_tracing` in fünf Teilbereiche zerlegt. Zusammen mit der Eingabe und Ausgabe ergeben sich folgende sieben Teilaufgaben wie sie auch in der Abb. *Der Datenfluss* wiederzufinden sind. Jeder dieser Bereiche lässt sich einem der nachfolgenden Unterkapiteln zuordnen. Zum Teil handelt es sich hierbei um die Implementierung eines speziellen Verfahrens, während das entsprechende Unterkapitel sich mit einem größeren Spektrum an Techniken befasst.

- Grabbing in Kapitel *Grabbing* erläutert.
- Thresholding in Kapitel *Segmentierung* erläutert.
- Flächenerkennung in Kapitel *Flächenerkennung* erläutert.
- Objektbewertung in Kapitel *Objektbewertung* erläutert.
- Subframing / Verfolgung in Kapitel *Subframing / Verfolgung* erläutert.

Die Beispielapplikation

- Hervorhebung ebenfalls in Kapitel Subframing / Verfolgung erläutert.
- Display Frame erläutert in Kapitel Grafisches-Fenster.

Abbildung 3-2. Datenflussdiagramm der Beispielapplikation



Data Dictionary:

- RGB-Bild

Ein Bild im RGB-Format.

- X0, Y0, X1, Y1

Es handelt sich um die Koordinaten der linken oberen Ecke (X0 und Y0) und der rechten unteren Ecke (X1 und Y1) des Subframe.

- Label-Bild

Die Beispielapplikation

Jede Pixel dieses Bildes ist über seinen Label-Wert einer Fläche zugeordnet. Weiter Informationen zu den Flächen befinden sich in der Labels-Tabelle.

- Labels-Tabelle

Eine Zeile diese Tabelle enthält die vorzeichenlosen Integer-Werte NEXT_INDEX, MENGE, X_MAX, X_MIN, Y_MAX, Y_MIN, MENGENDICHTE, CONNECTIONS und ROOT. Jede Zeile repräsentiert eine gefundene Fläche. Bei NEXT_INDEX und ROOT handelt es sich um Hilfsinformation, die während der Flächenerkennung benötigt wird. X_MAX, X_MIN, Y_MAX und Y_MIN beschreiben die Ausdehnung der Fläche. Die Werte MENGE, MENGENDICHTE und CONNECTIONS werden für die spätere Objektbewertung erzeugt. Für Details sei auf die Kapitel Flächenerkennung und Objektbewertung verwiesen.

- Eck-Pixel

Die Ecken eines Subframe werden in einem RGB-Bild farbig hervorgehoben, um zu zeigen, welches Objekt verfolgt wird.

- objekt_index

Index der Labels-Tabelle, der auf das Objekt verweist, welches verfolgt wird.

- X-Y-MIN-MAX-Werte

Enthält die vorzeichenlosen Integer-Werte X_MAX, X_MIN, Y_MAX, Y_MIN einer durch objekt_index gewählten Zeile der Labels-Tabelle.

- CONNECTIONS

Es werden die CONNECTIONS-Werte in der Labels-Tabelle ausgelesen.

- Vordergrund-Pixel

Pixel das vom Thresholding als Vordergrund-Pixel selektiert wurden.

Im Datenflussdiagramm (siehe Abb. Der Datenfluss) sind einige Funktionen und Speicher mehrlagig dargestellt. Alle so dargestellten Elemente können mehrfach instanziiert werden. Würden drei Objekte im Bild verfolgt werden, gäbe es jedes der so dargestellten Elemente dreifach. Daten fließen nur zwischen Elementen, die zu einem Objekt gehören.

Der Informationsfluss im Beispielprogramm beginnt in "Video IN" von dort wird durch die unter dem Begriff "Grabbing" zusammengefassten Methoden das Bild in einem Bereich des Hauptspeichers kopiert. Dieser Bereich wird im Datenflussdiagramm mit "RGB-Bild" bezeichnet.

Die mit "Thresholding" bezeichnete Funktion überprüft jeden Punkt des RGB-Bildes, der sich innerhalb des durch "X0, Y0, X1, Y1" aufgespannten Rechteckes befindet. Liegt ein solcher Punkt über der/(den) Threshold-Bedingung(-en), wird er als "Vordergrund-Pixel" an die Flächenerkennung weitergeleitet.

In der Flächenerkennung wird anhand des "Label-Bilds" überprüft, ob der Punkt zu einer Fläche gehört und dem entsprechend ein Wert in das "Label-Bild" eingetragen. Eine weitere Aufgabe der Flächenerkennung ist das Füllen und Aktualisieren der "Labels-Tabelle" mit den über die Flächen bekannten Werten.

Ist die Flächenerkennung abgeschlossen, wird in dem Bereich der Objektbewertung mit den "CONNECTIONS"-Werten der "Labels-Tabelle" geprüft, welches Objekt, zu verfolgen ist. Danach wird die Zeilennummer der "Labels-Tabelle" in der Variable "objekt_index" an den Bereich des "Subframing Verfolgung" weitergegeben. Im "Subframing Verfolgung" wird mit den "X-Y-MIN-MAX-Werten" die Größe des neuen Subframe in die Variablen "X0, Y0, X1, Y1" geschrieben.

Die Beispielapplikation

Die Aufgabe des "Ecken hervorhebens" ist es, mittels der "X0, Y0, X1, Y1" Informationen in das "RGB-Bild" Eckmarkierungen zu zeichnen. Durch diese Markierungen werden die Verfolgten Objekte hervorgehoben und die Größe des Subframes angezeigt.

Danach kann das RGB-Bild mit den unter "Display Frame" zusammengefassten Methoden in dem "Grafischen-Fenster" ausgegeben werden.

Übersicht der Implementierten Techniken

In den folgenden Unterkapiteln werden verschiedene Techniken vorgestellt, um eine spezielle Aufgabe zu lösen. Wird eine Technik in der Beispielapplikation implementiert, erfolgt ein Verweis auf die entsprechende Methode im Quellcode der Beispielapplikation. Die folgende Auflistung ist eine Übersicht der folgenden Unterkapitel und welche ihrer Techniken in der Beispielapplikation implementiert wurden. Für Begriffe, die einer weiteren Erläuterung bedürfen, sei auf das entsprechende Unterkapitel verwiesen.

- Grabbing

Dieses Kapitel wird komplett implementiert.

- Segmentierung

Um eine hohe Verarbeitungszeit zu gewährleisten, wird in der Beispielapplikation nur das Statisches Thresholding über die Summe der Farben und ein zweistufiges Thresholding für jede der drei Farben Rot, Grün und Blau implementiert.

- Flächenerkennung Flächenerkennung

Dieses Kapitel wurde in Form der 8-connectivity-Erkennung komplett implementiert.

- Objektbewertung

Hier wurde sowohl die Technik der Mengendichte als auch der Kreisflächenähnlichkeit implementiert. Aufgrund der Vorteile der Kreisflächenähnlichkeit wurde der Mengendichte-Bereich im Quellcode auskommentiert. Die Selektion erfolgt durch Wahl des Objektes mit dem grössten CONNECTIONS-Wert.

- Subframing / Verfolgung

Dieses Kapitel wurde komplett implementiert.

- Grafisches-Fenster

Hier wurde unter Verwendung von Xlib-Funktionen ein Grafisches-Fenster erzeugt, auf dem das überarbeitete RGB-Bild ausgegeben wird. Darüber hinaus wurde eine Ereignisbehandlung für Tasten und Mauseingaben realisiert.

Zurück

Objekterkennung/Objektverfolgung

Zum Anfang

Nach oben

Nach vorne

Grabbing

Grabbing

Mit Grabbing wird der Vorgang beschrieben, ein Bild aus einem Bildstrom oder Videostrom zu entnehmen. In dem Fall des Beispielprogrammes wird aus dem Strom, der von der *Frame-Grabber-Karte* empfangenen Bilder, so oft wie möglich ein Bild in den Speicher kopiert. Umgesetzt wird dieser Vorgang mit der *API Video4Linux* (kurz *v4l*). Diese Schnittstelle stellt dem Programmierer eine Anzahl von *System-Calls* zur Verfügung, mit denen sich der Grabvorgang durchführen und beeinflussen lässt. Bevor jedoch ein Frame eingefangen wird, muss ein Speicherbereich festgelegt werden, in dem der Frame abgelegt werden kann. Dieser kann z.B. durch die C-Befehle malloc oder mmap reserviert werden. Danach kann der Grabbing Vorgang eingeleitet werden. Um ein Frame einzufangen, müssen folgende Schritte durchgeführt werden.

- OPEN - Ein File Deskriptor, der auf das Video Device verweist wird erzeugt.
- INIT - Die Eigenschaften der Kamera sollen erfasst und die Kamera eingestellt werden.
- GRAB - Ein Frame wird eingefangen.
- SYNC - Es wird gewartet, bis das Bild eingefangen wurde.
- CLOSE - Der File Deskriptor wird wieder freigeben.

Hierbei ist zu beachten, dass nach dem Punkt "SYNC" sofort wieder der Punkt "GRAB" angesprungen wird, wenn es darum geht, mehrere Bilder in Folge zu erfassen. Erst wenn genug Bilder gegrabbt wurden, wird der letzte Punkt "CLOSE" abgearbeitet.

Leider reicht diese Technik noch nicht aus, um höhere Bildraten zu erreichen. Das kommt daher, dass die meiste Zeit darauf gewartet wird bis ein Bild einsynchronisiert ist. Da die meisten Video Devices über genug Speicher verfügen, um mehrere (meist zwei) Bilder aufnehmen zu können, benutzt man den Trick, zwei Grabvorgänge ineinander zu verschachteln. Weiter muss bedacht werden, dass es nicht ausreicht, ein Bild einzufangen, sondern dass ebenfalls eine Objekterkennung angewandt werden muss. So ergibt sich das Schema nach Abb. *Verschachteltes Grabben* für einen Grabvorgang wobei der Abschnitt "WORK" die eigentliche Objekterkennung repräsentiert.

Abbildung 3-3. Verschachteltes Grabben

```
OPEN();  
INIT();  
GRAB(Frame0);  
  
while (!Abbruch) {  
    GRAB(Frame1);  
    SYNC(Frame0);  
    WORK(Frame0);  
    GRAB(Frame0);  
    SYNC(Frame1);  
    WORK(Frame1);  
}  
CLOSE();
```

Der Pseudocode in Abb. Verschachteltes Grabben Version 2 erfüllt die selbe Funktion wie der Code in Abb. Verschachteltes Grabben.

Abbildung 3-4. Verschachteltes Grabben Version 2

```
FRAME=0;

OPEN();
INIT();
GRAB(FRAME);

while (!Abbruch){
    GRAB(!FRAME);
    SYNC(FRAME);
    WORK(FRAME);
    FRAME = !FRAME;
}
CLOSE();
```

Gerade in dieser zweiten Version (Abb. Verschachteltes Grabben Version 2) erkennt man die Vorteile des verschachtelten Grabbens. Zum besseren Verständnis werden im Folgenden zwei Schleifendurchläufe der While-Schleife erläutert.

In diesem Durchlauf sei FRAME=0. In der ersten Anweisung der Schleife wird der Grabvorgang für den FRAME=1 aktiviert. Da dieser im Hintergrund auf der Frame-Grabber-Karte abläuft, bleibt Zeit und Rechenleistung, um den Frame-0 zu synchronisieren (SYNC Anweisung) und nach Objekten zu durchsuchen (WORK Anweisung). Am Ende des Schleifendurchgangs wird FRAME von 0 auf 1 gesetzt (getoggelt). Im nächsten Durchlauf (FRAME=1) wird der Grabvorgang für Frame-0 aktiviert und während er im Hintergrund abläuft, kann Frame-1 synchronisiert und nach Objekten durchsucht werden. Am Ende des Schleifendurchgangs wird FRAME von 1 auf 0 gesetzt (getoggelt). So werden die Pausen zwischen Grabben und Synchronisieren für die Bearbeitung des jeweils anderen Frame ausgenutzt.

In der Beispielapplikation wird der OPEN-Vorgang von der Methode `video_in::grab_open` und der CLOSE-Vorgang von der Methode `video_in::grab_close` übernommen. Für das Einfangen der Frames ist die Methode `video_in::grab_pix`, die wiederum die Methode `video_in::grab_frame` und `video_in::grab_sync` aufruft, zuständig.

In der Beispielapplikation wird die Technik des verschachtelten Grabbens verwendet.

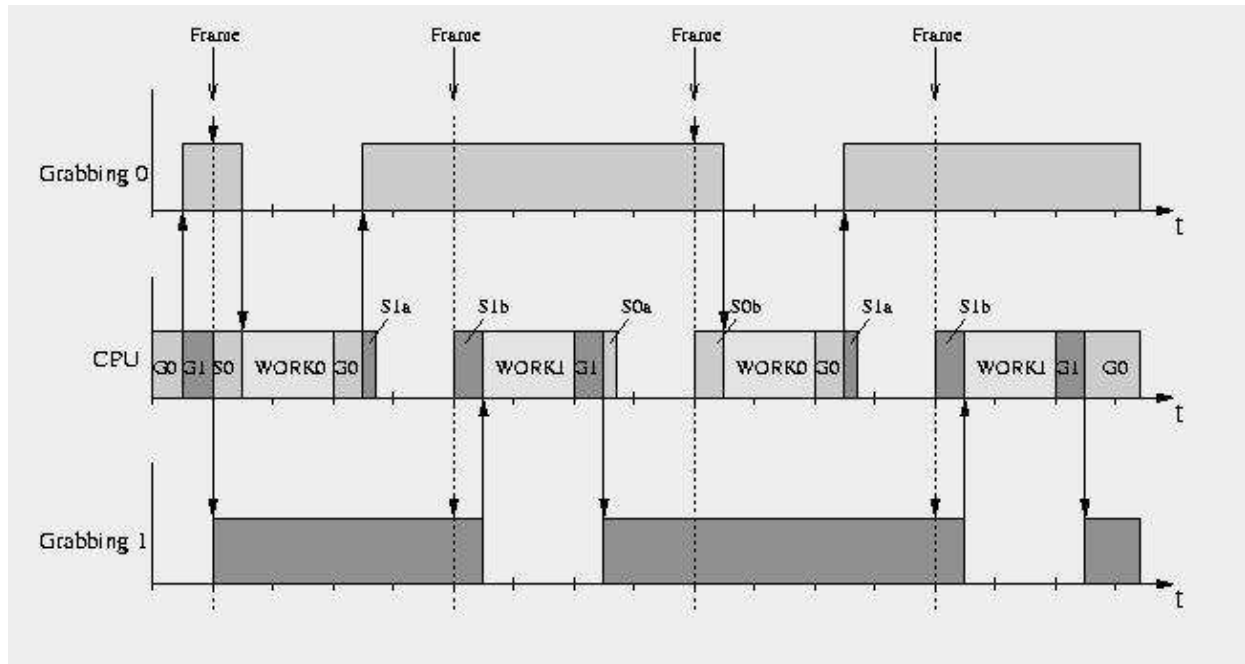
Interessant ist auch die Betrachtung einer vereinfachten Darstellung der Rechenauslastung für eine kleine Reihe von Grabvorgängen (siehe Abb. Die Auslastung beim verschachtelten Grabben). Vereinfacht wurde hier beispielsweise durch das Weglassen von Prozessen, die mit unserer Grabapplikation konkurrieren.

Da ein Frame nach einem Grab-Auftrag so bald wie möglich von der Frame-Grabber-Karte in ihren Speicherbereich kopiert wird, sind in der Abbildung zwei weitere Zeitachsen angegeben (Grabbing 0 und 1) auf denen dargestellt wird, von wann bis wann die Karte auf einen Frame wartet, und wann der Frame in den Speicher der Karte übernommen wird. Fallen die Captureanfragen zweier Puffer zusammen, wird diejenige zuerst bearbeitet, die zuerst gestellt wurde. Erst durch das SYNC wird die Captureanweisung zurückgenommen, der Puffer kopiert und wieder freigegeben. Eine SYNC-Anweisung blockiert im Regelfall so lange bis für sie ein Frame vorhanden ist. In der Abbildung sieht man, dass eine SYNC-Anweisung für die kein Frame vorhanden ist unterbrochen wird (z.B. S1a) und erst nach Eintreffen des Frames fortgesetzt wird (S1b).

Grabbing

Auf der CPU Zeitachse konkurrieren sechs Prozesse um die CPU Leistung: die beiden Grab-Vorgänge G0, G1, die dazugehörigen Sync-Vorgänge S0, S1 und die Objekterkennung WORK0, WORK1, die einmal Frame-0 und ein andermal Frame-1 bearbeitet. In der Abbildung lässt sich erkennen, dass während die Frame-Grabber-Karte z.B. Frame0 einfängt, die Objekterkennung von Frame1 durchgeführt wird, und umgekehrt.

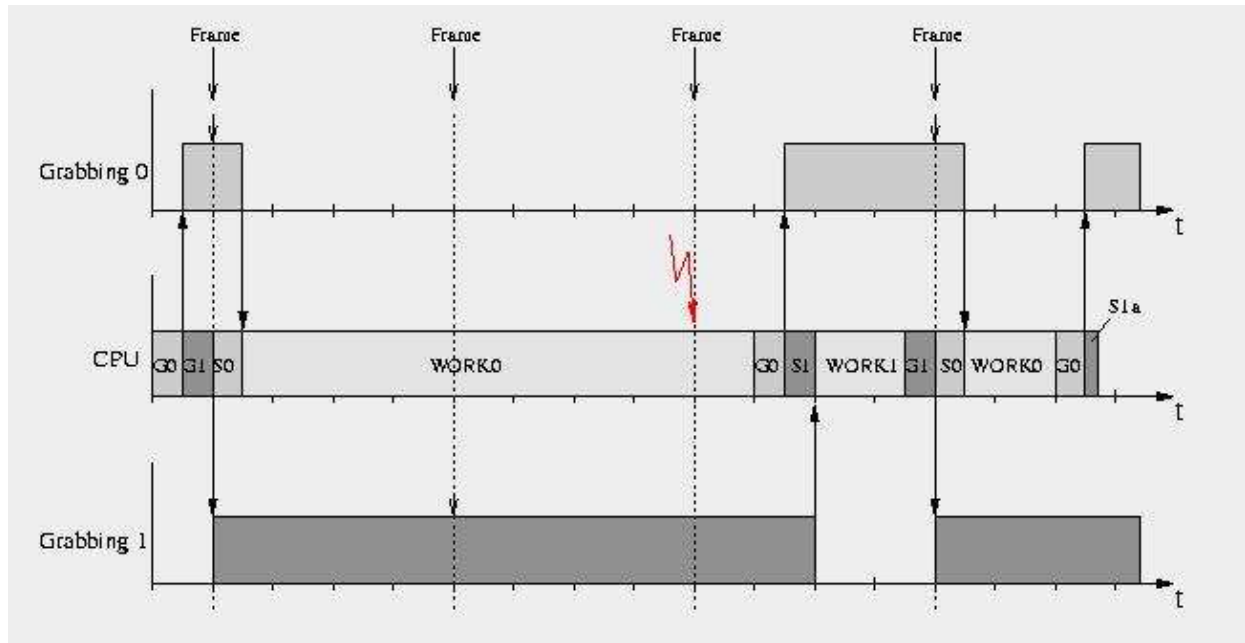
Abbildung 3-5. Die Auslastung beim verschachtelten Grabben



Im folgenden Fall einer Deadlinverletzung (siehe Abb. Die Auslastung beim Verschachtelten Grabben mit Deadline-Verletzung) benötigt der Prozess WORK0 für die Objekterkennung so viel Zeit, dass der anschließende Grabvorgang G0 erst gestartet wird, nachdem er (G0) den für ihn bestimmten Frame verpasst hat. Es handelt sich hierbei um eine Verletzung der 1. Echtzeitbedingung - Pünktlichkeit, die bewirkt, dass ein Frame nicht gegrabbt wird und für die Objekterkennung verloren geht. Es handelt sich hierbei um das Nichteinhalten einer weichen Echtzeitbedingung. Würde hingegen die Objekterkennung immer so viel Rechenleistung beanspruchen, dass die Gesamtrechenleistung nicht ausreicht, wäre von einer Verletzung der 2. Echtzeitbedingung - Auslastung die Rede. Als Folge würde nur noch jeder zweite oder n-te Frame verarbeitet werden.

Abbildung 3-6. Die Auslastung beim Verschachtelten Grabben mit Deadline-Verletzung

Grabbing



Eine Einführung in die Verwendung der *Video4Linux API* findet man unter der Quelle [Cox2000].

[Zurück](#)

Die Beispielapplikation

[Zum Anfang](#)

[Nach oben](#)

[Nach vorne](#)

Segmentierung

Segmentierung

Wurde das Bild in den Arbeitsspeicher kopiert, kann mit der Segmentierung begonnen werden. Die Aufgabe der Segmentierung ist es, das Bild in unterschiedliche Bereiche zu unterteilen, die alle die gleichen Kriterien erfüllen. Oft wird das Bild in zwei Bereiche unterteilt. Zum einen in den Vordergrund, der die interessanten Elemente enthält und zum anderen in den *Hintergrund*, der Informationen enthält, die für die weitere Bearbeitung entfernt werden müssen. Recht einfach lässt sich dies erfüllen, wenn als Kriterium ein gemeinsamer Grauwert oder eine gemeinsame Farbe gewählt wird. Komplizierter wird es, wenn die gleichförmige Form oder Struktur als Kriterium dient.

Weiter wird unterschieden, ob das Kriterium bekannt ist, z.B. eine bestimmte Farbe oder eine bestimmte Form hat, oder ob es durch die Objekterkennung ermittelt werden muss. Dieses Kapitel befasst sich hauptsächlich mit der Segmentierung nach Farbe oder Grauwert, da eine Unterscheidung nach Form oder Struktur für eine Echtzeit Objekterkennung zu rechenzeitaufwendig wäre. Für Informationen zu diesen Themen sei auf die Quellen [FiPeWaWo2000] und [YoGeV11999] verwiesen.

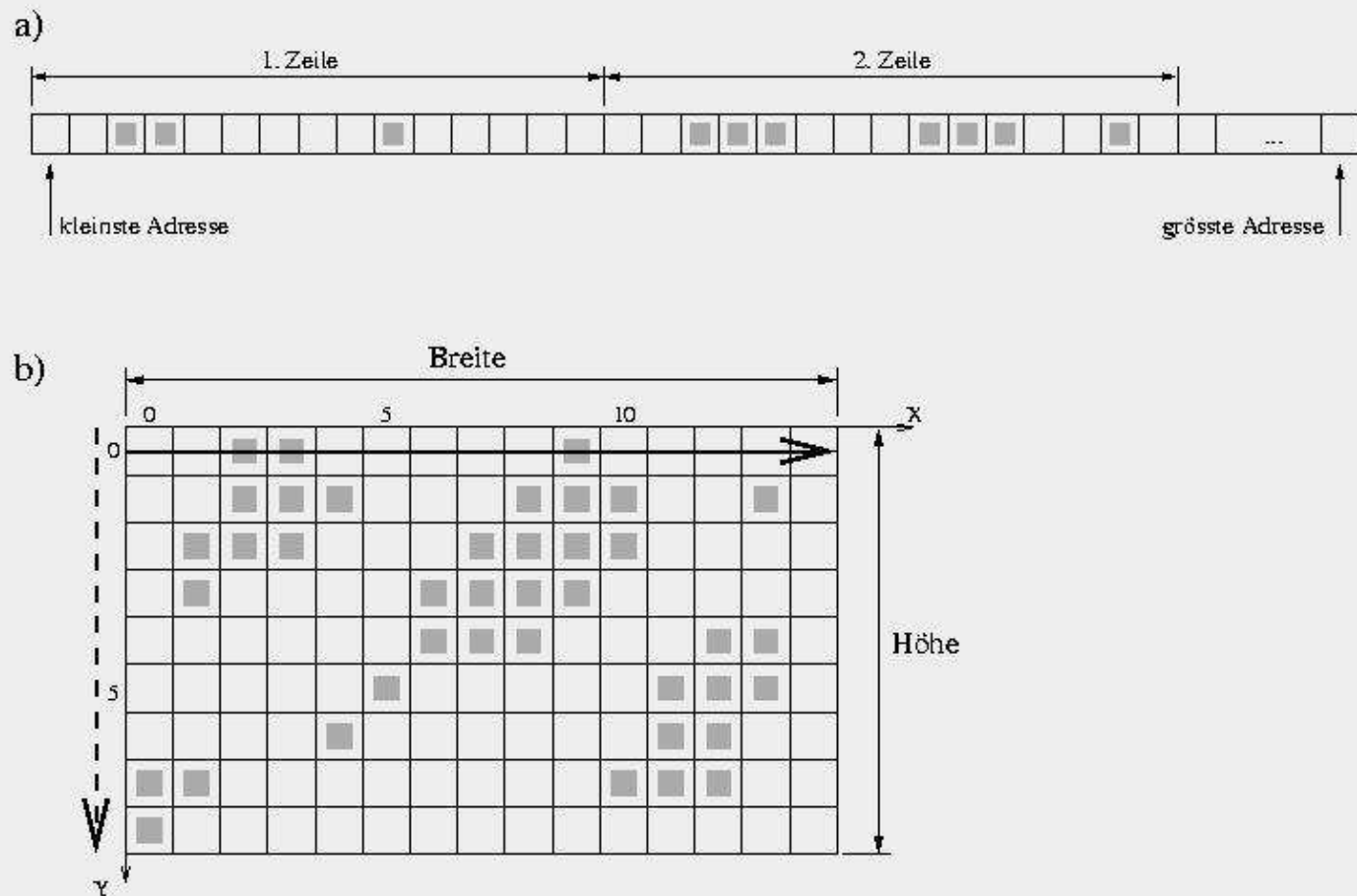
Zum Thema Segmentierung gibt es keine Universallösung. Vielmehr ist die verwendete Technik stark von der Aufgabenstellung und dem zu analysierenden Objekt abhängig. Auch gibt es für dieses Gebiet keine kompromisslose, perfekte Lösung.

Verarbeitungsrichtung

Wie in Abb. *Die Verarbeitungsrichtung der Segmentierung und der Flächenerkennung - Teilbild b)* durch die beiden großen Pfeile angedeutet, ist die normale Verarbeitungsrichtung beim Segmentieren und Flächenerkennen zeilenweise von links oben nach rechts unten. Das heißt, dass erst die oberste Zeile von links nach rechts abgearbeitet wird und anschließend die darunterliegenden Zeilen, bis das gesamte Bild bearbeitet ist. Die Ursache dieser Richtung ist in Abb. *Die Verarbeitungsrichtung der Segmentierung und der Flächenerkennung - Teilbild a)* zu erkennen. Wenn ein Bild sich im Arbeitsspeicher befindet, belegt es einen kontinuierlichen Speicherbereich. Wird jetzt das Bild bearbeitet, wird der Speicherbereich üblicherweise, beginnend an der kleinsten Adresse bis zur höchsten Adresse, bearbeitet.

Abbildung 3-7. Die Verarbeitungsrichtung der Segmentierung und der Flächenerkennung

Segmentierung



Muss diesem Bild bei der Bearbeitung Eigenschaften wie eine Höhe oder eine Breite bzw. ein Koordinatensystem zugeordnet werden (Abb. Die Verarbeitungsrichtung der Segmentierung und der Flächenerkennung - Teilbild b)), so geschieht dies durch eine Zähllogik in der Software oder durch eine Ganzzahl-Division der aktuellen Pixelposition durch die Breite des Bildes. Hierbei ergibt der ganzzwertige Anteil des Quotienten die aktuelle Position auf der Abszisse (X-Achse), und der Rest die Position auf der Ordinate (Y-Achse). In vielen Programmiersprachen lässt sich dies trivial durch eine Division und Modulo Operation implementieren.

Weiter ist zu beachten, dass der Nullpunkt des Koordinatensystems sich in der linken oberen Ecke befindet und die Ordinate (Y-Achse), im Gegensatz zu einem sonst üblichen Koordinatensystem, mit ihren positiven Werten nach unten weist. Das Pixel am Ursprung dieses Systems hat die Koordinaten $X=0$ und $Y=0$.

Statisches Thresholding

In der einfachsten Implementierung werden alle Pixel eines aus Grauwerten bestehenden Bildes gelesen und mit einem festen Schwellwert / Threshold verglichen. Ist der Grau-Wert des Pixel kleiner oder gleich dem Schwellwert, wird das Pixel dem Hintergrund zugeordnet und beispielsweise auf Schwarz gesetzt. Ist der Wert des Pixels größer als der Schwellwert, wird das Pixel dem Vordergrund zugeordnet und z.B. auf Weiß gesetzt. Je nach Anwendung kann Hintergrund und Vordergrund auch anders herum den Schwellwerten zugeordnet sein. Bei einem Farbbild im RGB-Format wird diese Technik für jede der drei Farben, oder für die Summe der drei Farben angewandt. Im Folgenden wird dies anhand von zwei Beispielen erläutert.

Segmentierung

Ein Verfahren ist, die Farben erst aufzuaddieren und dann den Threshold-Filter auf ihre Summe anzuwenden. Natürlich muss der Schwellwert entsprechend groß gewählt werden, da die einzelnen Farbwerte summiert werden. (Siehe Abb. Threshold über die Summe der Farben und den Beispielcode in Abb. Beispielcode: Threshold über die Summe der Farben)

Eine entsprechende Implementierung befindet sich in der Methode `o_tracing::threshold` der Beispielapplikation. Durch den Aufruf der Methode `o_tracing::set_total_threshold` kann zuvor das Thresholdniveau eingestellt werden.

Abbildung 3-8. Threshold über die Summe der Farben

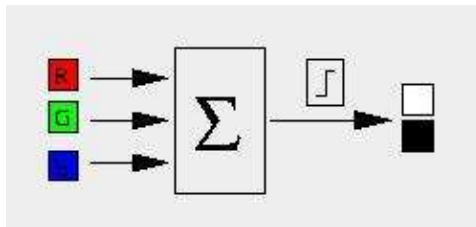


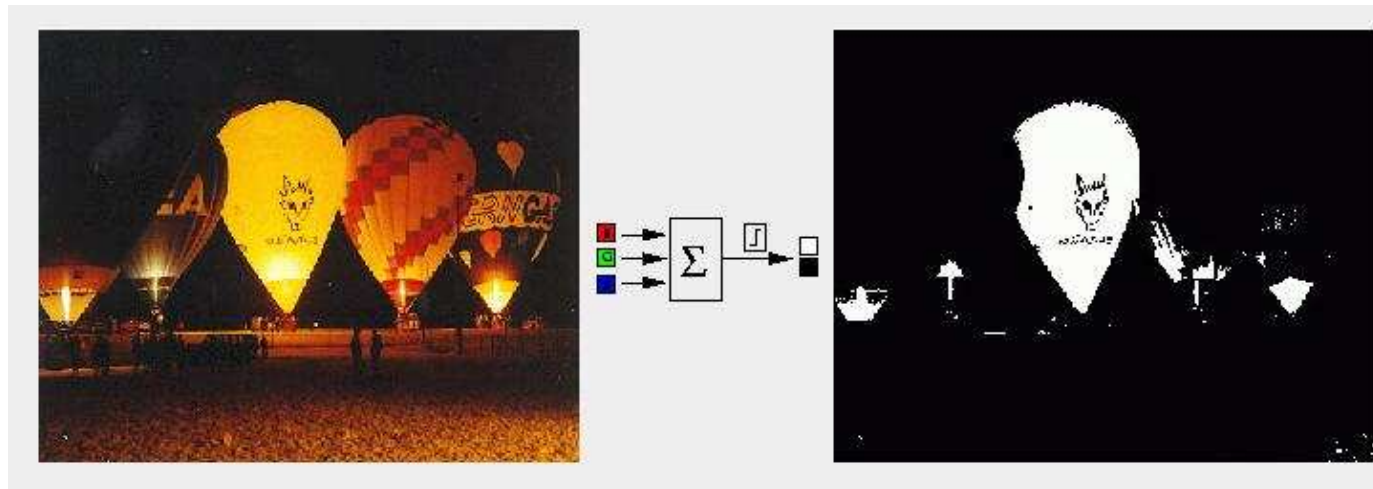
Abbildung 3-9. Beispielcode: Threshold über die Summe der Farben

```
if (Q_Punkt[rot] + Q_Punkt[gruen] + Q_Punkt[blau] <= SCHWELLWERT){  
    Z_Punkt=0;  
} else {Z_Punkt=255;  
}
```

Wendet man einen solchen Filter auf ein Bild an (siehe Abb. Summierender Threshold auf ein Bild angewendet), erkennt man deutlich, dass die hellsten Objekte dem Vordergrund zugeordnet werden und die Farbe Weiß erhalten, während der Rest zu schwarzem Hintergrund gefiltert wird. Der Schwellwert wurde bewusst so gewählt, dass nur der hellste Ballon im Bild als dominierendes Vordergrundelement erhalten bleibt.

Abbildung 3-10. Summierender Threshold auf ein Bild angewendet

Segmentierung



Eine andere Art den Threshold-Filter zu wählen, ist jede Farbe einzeln darauf zu testen, ob sie innerhalb eines definierten Wertebereichs liegen. Das Ergebnis dieses Vergleiches wird dann *logisch UND-Verknüpft*, so dass nur, wenn alle drei Farbwerte innerhalb der Wertebereiche liegen, das Ausgangsbild auf weiß gesetzt wird. Man spricht hierbei auch von einem zweistufigen Threshold. (Siehe Abb. Zweistufiges Threshold für jede der drei Farben und den Beispielcode in Abb. Beispielcode: Zweistufiges Threshold für jede der drei Farben.)

Die entsprechende Implementierung befindet sich in der Methode `o_tracing::threshold` der Beispielapplikation. Durch den Aufruf der Methode `o_tracing::set_rgb_threshold` können zuvor die Wertebereiche der drei Farben eingestellt werden.

Abbildung 3-11. Zweistufiges Threshold für jede der drei Farben

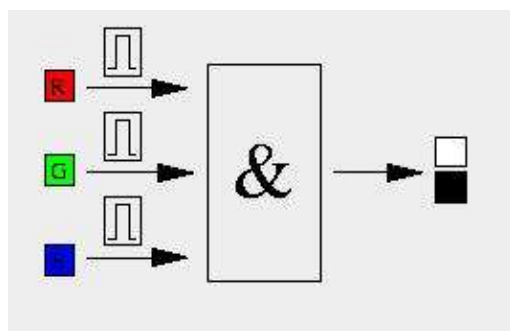


Abbildung 3-12. Beispielcode: Zweistufiges Threshold für jede der drei Farben

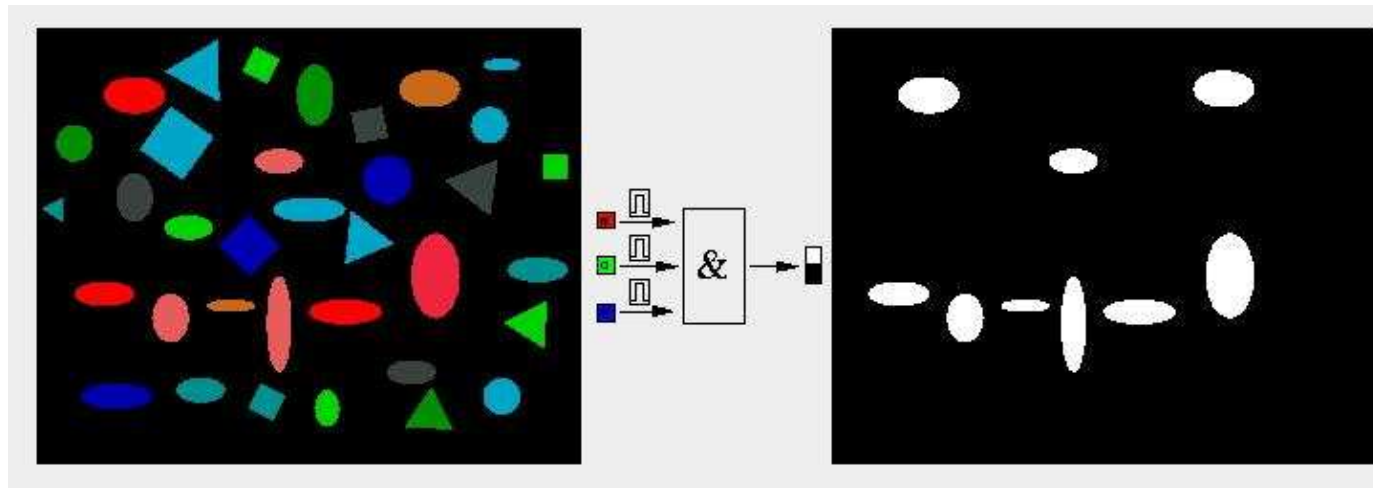
```
if (      (ROT_MAX   >= Q_Punkt[rot]   > ROT_MIN   )
    && (GRUEN_MAX  >= Q_Punkt[gruen]  > GRUEN_MIN)
    && (BLAU_MAX   >= Q_Punkt[blau]   > BLAU_MIN  ) ){
    Z_Punkt=0;
}
```

Segmentierung

```
else {Z_Punkt=255;  
}
```

In der folgenden Abbildung ist die Wirkungsweise eines solchen Filters dargestellt. Wird ein solches Verfahren auf ein Bild angewendet, ist zu erkennen, dass hier nur Elemente mit bestimmten Farbanteilen den Filter passieren. In dem hier gezeigten Fall verbleiben nur Farben mit einem hohen Rotanteil (siehe Abb. Wirkungsweise eines zweistufigen Threshold auf jede der drei Farben).

Abbildung 3-13. Wirkungsweise eines zweistufigen Threshold auf jede der drei Farben

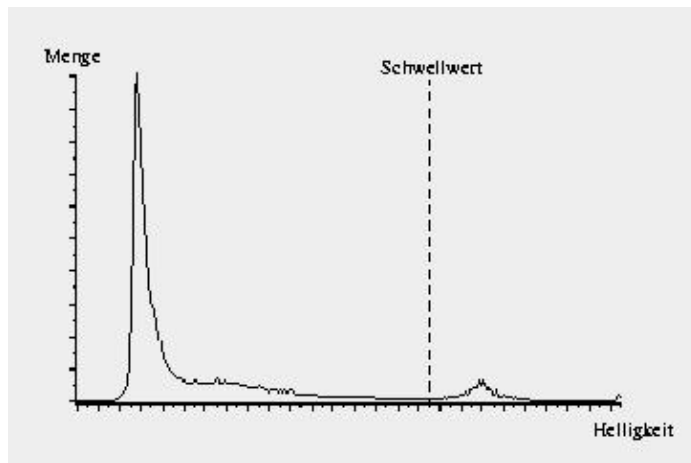


Schwellwertbestimmung mit Hilfe von Histogrammen

Während bis jetzt die Schwellwerte zur Trennung zwischen Vordergrund und Hintergrund fest bestimmt wurden, ist es gerade in Umgebungen mit veränderlichen Lichtverhältnissen oft nötig, diese durch die Betrachtung des Helligkeits-Histogramms automatisch durchzuführen. Unter Helligkeits-Histogramm versteht man, dass ein Bild, welches nur aus Grauwerten besteht, pixelweise durchlaufen wird. Hierbei wird eine Tabelle angelegt, die angibt, welcher Grauwert mit welcher Anzahl im Bild auftritt. Trägt man nun auf der Abszisse (X-Achse) die möglichen Grauwerte und auf der Ordinate (Y-Achse) die Anzahl ihres Vorkommen im Bild ein, so erhält man ein Helligkeits-Histogramm (siehe Abb. Grauwert-Histogramm des Ballonbildes). Hierfür wurde das Ballonbild (siehe linkes Bild von Abb. Summierender Threshold auf ein Bild angewendet) in Grauwerte gewandelt und daraus das Grauwert-Histogramm erzeugt. Für ein Grauwertbild, bei dem die Helligkeit jedes Bildpunktes durch 8 Bit beschrieben wird, kann ein Pixel einen von 256 ($2^8=256$) Grauwerten annehmen. Ein daraus ermitteltes Histogramm würde demnach 256 diskrete Werte umfassen. Helligkeitswerte, die nicht im Bild vorkommen, erhalten im Histogramm den Wert 0.

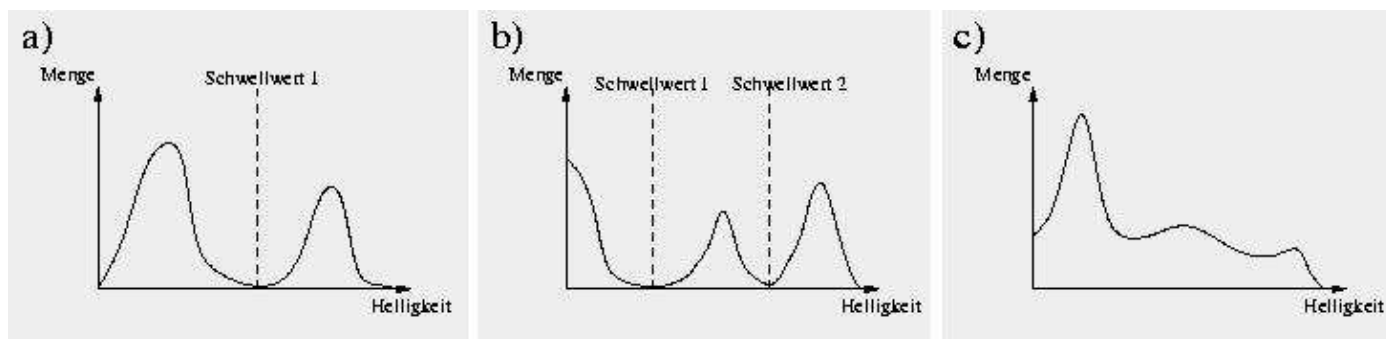
Abbildung 3-14. Grauwert-Histogramm des Ballonbildes

Segmentierung



Erkennbar ist, dass das Diagramm zwei Maxima hat. Links die dunklen Hintergrund-Farben und rechts die weniger hellen Elemente im Bild. Ein guter Punkt, um den Schwellwert zu plazieren, ist immer zwischen zwei solchen Spitzen. Veranschaulichen lässt sich dieses durch Abb. Bewertung von Histogrammen - Teilbild a). Soll ein zweistufiges Thresholding verwendet werden, müssen die Schwellwerte wie in Abb. Bewertung von Histogrammen - Teilbild b) gewählt werden. Das Histogramm des Bildes muss sich in mindestens drei Maxima unterteilen lassen. Verfügt das Histogramm eines Bildes über nur unzureichend ausgeprägte Maxima, wird es sehr schwierig, Objekte anhand ihrer Helligkeit zu erkennen, da sie sich nicht ausreichend von der Umgebung bzw. dem Hintergrund abheben. In Abb. Bewertung von Histogrammen - Teilbild c) ist ein solches Histogramm dargestellt.

Abbildung 3-15. Bewertung von Histogrammen



In dieser Arbeit werden nur Grauwert-Histogramme behandelt. Um von einem Farbbild ein Histogramm zu erzeugen, wird für jede der drei Farben, Rot, Grün und Blau, ein eigenes Histogramm erzeugt. Jedes der Histogramme kann nun einzeln bewertet werden. So lassen sich für alle drei Farben entsprechende Schwellwerte bilden. Erfolgt dann die Filterung, so wird für einen Punkt jede Farbe getrennt betrachtet und durch eine logische Verknüpfung der Teil-Ergebnisse das Ausgangsergebnis gebildet.

Automatische Schwellwertbestimmung

Betrachtet man ein Histogramm nach den Regeln der Mengenlehre, ist b die Menge, der auf der Abszisse angegebenen Grauwerte $\{b = 0, 1, \dots, 2^B - 1\}$, deren Höchstwert, wie in der Computertechnik üblich, auf der Basis 2 beruht. Ein dafür üblicher Wert wäre $2^{16} - 1 = 256 - 1 = 255$. Die auf der Ordinate aufgetragenen Anzahl

Segmentierung

wird mit der Menge $\{h[b]\}$ bezeichnet.

Ein Algorithmus, um den Schwellwert automatisch zu ermitteln, ist beispielsweise der iterative *Isodata Algorithmus* von Ridler und Calvard.

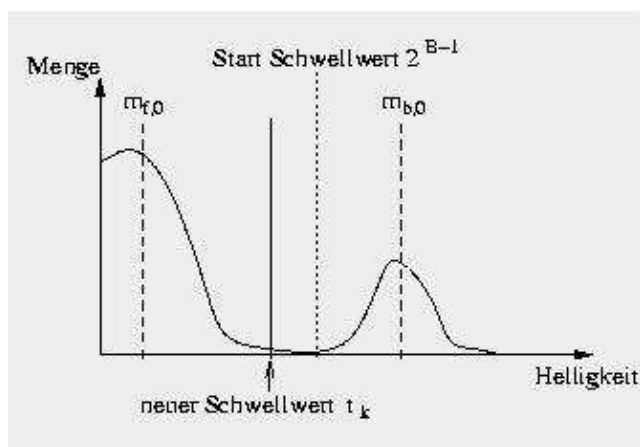
Zu Beginn wird das Histogramm durch einen Startschwellwert $t_0=2^{B-1}$ in zwei Teile unterteilt. Von beiden dieser Teile wird nun der Helligkeitsmittelwert berechnet ($m_{f,0}$). Diesen Helligkeitsmittelwert erhält man durch Bildung einer Summe über alle Multiplikation der Helligkeitswerte mit denen ihnen zugeordneten Mengen, geteilt durch die Summe aller Mengen. Siehe hierfür Abb. Formel zur Errechnung eines Mittelwertes.

Abbildung 3-16. Formel zur Errechnung eines Mittelwertes

$$\text{Mittelwert} = \frac{\sum_{i=0}^n (\text{Helligkeit}_i * \text{Menge}(\text{Helligkeit}_i))}{\sum_{i=0}^n \text{Menge}(\text{Helligkeit}_i)}$$

Durch das Mitteln dieser beiden Helligkeitswerte ergibt sich nun ein neuer Thresholdwert t_1 , mit dem dieser Algorithmus erneut durchlaufen wird wie in Abb. Veranschaulichung des Isodata Algorithmus zu erkennen ist.

Abbildung 3-17. Veranschaulichung des Isodata Algorithmus



Dieser Vorgang wird wiederholt bis der Schwellwert aus dem vorhergehenden Durchgang sich nicht mehr wesentlich vom neu errechneten Schwellwert unterscheidet, oder bis eine maximale Anzahl von Durchläufen erreicht wird. Abb. Isodata Algorithmus zur Bestimmung eines einfachen Schwellwertes zeigt die entsprechende Formel.

Abbildung 3-18. Isodata Algorithmus zur Bestimmung eines einfachen Schwellwertes

$$t_k = \frac{(m_{t,k-1} + m_{b,k-1})}{2} \quad \text{until} \quad t_k = t_{k-1}$$

Sollen für ein Bild ein mehrstufiger Threshold verwendet werden, muss eine erweiterte Form des Isodata Algorithmus verwendet werden. Hierfür und für detailliertere Angaben zum Algorithmus sei auf die Quelle [RiCa1978] verwiesen.

Adaptives Thresholding

Ist das Bild ungleichmäßig ausgeleuchtet, reicht ein einfacher Threshold oft nicht aus, um ein ausreichendes Ergebnis zu erzielen. In diesem Fall muss auf eine adaptive Thresholdtechnik zurückgegriffen werden. Die Verfahren des adaptiven Thresholding basieren alle auf dem Prinzip, dass kleine Flächen gleichmäßiger ausgeleuchtet sind als eine große Fläche.

Eine von Chow und Kaneko entwickelte Technik zerlegt ein Bild in mehrere sich nicht überschneidende Bereiche. Für jeden dieser Bereiche wird ein Histogramm erzeugt und ein Schwellwert bestimmt. Die einzelnen Schwellwerte werden zusammengefügt und interpoliert, so dass sich für jeden Punkt der gesamten Bilder ein Thresholdwert ergibt. Ein Nachteil dieser Technik ist der hohe Bedarf an Rechenleistung.

Eine andere Methode, die weniger Rechenleistung beansprucht, basiert darauf, die umliegenden Pixel für die Bestimmung des Schwellwertes heranzuziehen. So wird aus einer gewissen Anzahl von räumlich nahen Punkten ein Mittelwert gebildet, von dem sich das gerade betrachtete Pixel um eine feste Differenz abheben muss, um als Vordergrund-Pixel eingestuft zu werden. Wichtige feste Faktoren, die diesen Vorgang beeinflussen, sind die Wahl der Feldgröße aus der ein Mittelwert gebildet wird und die Differenz, um die sich das Pixel abheben muss.

Bei beiden Techniken ist es wichtig, die Feldgröße so zu wählen, dass sich sowohl Vordergrund als auch Hintergrund-Pixel innerhalb dieser Bereiche befinden. Ist die Feldgröße zu gering gewählt, geschieht es, dass beispielsweise das Zentrum einer ausgedehnten, hellen, dem Vordergrund zugedachten Fläche zu Hintergrund wird, da sich das Thresholdniveau zu weit anhebt. Genauso kann z.B. das Thresholdniveau in dunklen Hintergrund-Bereichen so weit absinken, dass leichte Helligkeitsschwankungen im Bild schon zu einer fälschlichen Erkennung von Vordergrundpixeln führt. Gerade diesem zuletztgenannten Effekt wirkt das Verwenden der Differenz entgegen. Wird die Feldgröße hingegen zu groß gewählt, so geht der Vorteil des Adaptiven Thresholding verloren und ungleichmäßige Beleuchtungseinflüsse können nicht mehr ausgeglichen werden.

Andere Arten der Segmentierung

Neben der Segmentierung nach Farben und Helligkeiten ist die Kantenerkennung eine weit verbreitete Technik. Da bei diesem Verfahren ein hohes Maß an Rechenleistung benötigt wird, eignet es sich nicht zur Anwendung einer Echtzeit Objekterkennung bei der durch hohe Bildraten nur wenig Rechenzeit verfügbar ist.

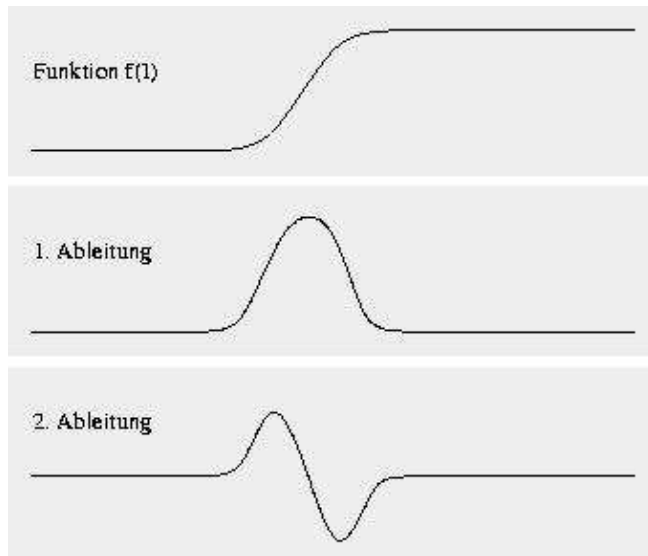
Gerade bei der Kantenerkennung ist es empfehlenswert, das Bild vorab mit Filtern gegen Rauschen zu bearbeiten, da diese Techniken sehr anfällig für solche Störungen ist.

Untersucht man die Kante eines Bildes (I) in einer Orthogonal (90 Grad) zur Kante verlaufenden Betrachtungsweise, handelt es sich dabei um den Anstieg oder das Abfallen eines Helligkeitswertes (siehe

Segmentierung

Abb. Eine Kante mit ihrer ersten und zweiten Ableitung). Zum Erkennen der Kante werden die beiden Ableitungen der Funktion $f(I)$ betrachtet. Man erkennt, dass die erste Ableitung an der Stelle der Kante ihr Maximum hat, und dass in der zweiten Ableitung an dieser Stelle ein Nulldurchgang liegt.

Abbildung 3-19. Eine Kante mit ihrer ersten und zweiten Ableitung



Anstatt die erste und zweite Ableitung für ein Bild zu berechnen, wird eine Technik, die sich *Convolution* (deutsch Verwindung) nennt, verwendet. So wird die erste Ableitung durch einen $[-1 \ 1]$ *Kernel* (Erläuterung erfolgt im Folgenden) und die zweite Ableitung durch einen $[1 \ -2 \ 1]$ *Kernel* gebildet.

Convolution ist eine Technik, um zwei Felder mit gleicher Dimensionalität, aber normalerweise unterschiedlicher Größe, zu addieren. Mit dieser Technik lassen sich in der Bildverarbeitung Ausgangs-Bildpunkte errechnen, deren Werte eine lineare Kombination von verschiedenen Eingangs-Pixelwerten darstellt.

In der Bildverarbeitung ist in der Regel das größere der beiden Felder das Graustufen-Bild, das bearbeitet werden soll. Das kleinere der beiden wird *Kernel* genannt und beschreibt die Operation, welche angewendet werden soll (siehe Abb. Veranschaulichung eines Image und eines Kernel) .

Abbildung 3-20. Veranschaulichung eines Image und eines Kernel

Segmentierung

Image								Kernel		
11,1	11,2	11,3	11,4	11,5	11,6	11,7	11,8	K1,1	K1,2	K1,3
12,1	12,2	12,3	12,4	12,5	12,6	12,7	12,8	K2,1	K2,2	K2,3
13,1	13,2	13,3	13,4	13,5	13,6	13,7	13,8			
14,1	14,2	14,3	14,4	14,5	14,6	14,7	14,8			
15,1	15,2	15,3	15,4	15,5	15,6	15,7	15,8			
16,1	16,2	16,3	16,4	16,5	16,6	16,7	16,8			
17,1	17,2	17,3	17,4	17,5	17,6	17,7	17,8			

Um die Convolution zu berechnen, durchläuft der *Kernel* das Image zeilenweise von links oben nach rechts unten. Bei dieser Technik wird für jede Position, an der der *Kernel* sich komplett auf dem Image befindet, ein Punkt des Ausgangsbildes berechnet. Hierfür wird jeder Kernel-Punkt mit dem darunterliegenden Image-Punkt multipliziert. Der Ausgabe-Punkt ergibt sich dann aus der Summe dieser Multiplikationen. Für das diagonal schraffierte Feld (siehe Abb. Veranschaulichung eines Image und eines Kernel) ist die Berechnung im Folgenden dargestellt (Abb. Berechnung einer Convolution).

Abbildung 3-21. Berechnung einer Convolution

$$O_{3,3} = I_{3,3} K_{1,1} + I_{3,4} K_{1,2} + I_{3,5} K_{1,3} + I_{4,3} K_{2,1} + I_{4,4} K_{2,2} + I_{4,5} K_{2,3}$$

Allgemein gilt als Berechnungsvorschrift die Formel: Formel zur Berechnung einer Convolution. Dabei sei M die Zeilen- und N die Spaltenanzahl des Images sowie m und n die Zeilen- und Spaltenanzahl des *Kernel*.

Abbildung 3-22. Formel zur Berechnung einer Convolution

$$O(i,j) = \sum_{k=1}^{M-m+1} \sum_{l=1}^{N-n+1} I(i+k-1, j+l-1) K(k,l)$$

Ein Nachteil dieser Technik ist, dass das neu produzierte Bild nur die Größe $M - m + 1$ und $N - n + 1$ hat und somit kleiner als das Ausgangsbild ist.

Die gängigsten *Kernel* für die Kantenerkennung, die auf dem Prinzip der ersten Ableitung basieren sind *Sobel* und *Roberts Cross*.

Ein *Kernel*, der auf der zweiten Ableitung basiert ist von *Marr*. Bei diesem Filter wird die zweite Ableitung mit einer angenäherten *Laplace Transformation* gebildet. Der Vorteil dieses Verfahren ist, dass in einem

Segmentierung

Durchlauf alle Kanten ermittelt werden, unabhängig wie diese im Bild ausgerichtet sind. Die Filter *Sobel* und *Roberts Cross* müssen rotiert und erneut über das Bild geschickt werden um die horizontalen, vertikalen und diagonalen Kanten gleichstark zu bewerten. Ein Nachteil ist, dass dieser Filter durch das Finden von Nulldurchgängen der zweiten Ableitung sehr stark auf *Rauschen* reagiert. In Abb. *Typische Kernel* sind einige *Kernel* dargestellt.

Abbildung 3-23. Typische Kernel

Sobel	Roberts Cross	Marr																						
2 Durchläufe	2 Durchläufe	1 Durchlauf																						
<table><tr><td>-1</td><td>0</td><td>+1</td></tr><tr><td>-2</td><td>0</td><td>+2</td></tr><tr><td>-1</td><td>0</td><td>+1</td></tr></table>	-1	0	+1	-2	0	+2	-1	0	+1	<table><tr><td>+1</td><td>0</td></tr><tr><td>0</td><td>-1</td></tr></table>	+1	0	0	-1	<table><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>-4</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr></table>	0	1	0	1	-4	1	0	1	0
-1	0	+1																						
-2	0	+2																						
-1	0	+1																						
+1	0																							
0	-1																							
0	1	0																						
1	-4	1																						
0	1	0																						
90 Grad rotiert	90 Grad rotiert																							
<table><tr><td>+1</td><td>+2</td><td>+1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>-1</td><td>-2</td><td>-1</td></tr></table>	+1	+2	+1	0	0	0	-1	-2	-1	<table><tr><td>0</td><td>+1</td></tr><tr><td>-1</td><td>0</td></tr></table>	0	+1	-1	0										
+1	+2	+1																						
0	0	0																						
-1	-2	-1																						
0	+1																							
-1	0																							

Nachdem das Bild mit Hilfe der Convolution-Technik gefiltert wurde, kann man z.B. durch einen einfachen Threshold alles bis auf die stärksten Kanten entfernen. Wichtig ist, dass die Kanten möglichst keine Lücken aufweisen und das zu erkennende Objekt einschließen.

Für detailliertere Informationen was z.B. die verschiedenen *Kernel* oder die Detektion von durch Kanten eingeschlossene Körper betrifft, sei auf die Quellen [FiPeWaWo2000] und [YoGeV11999] verwiesen.

Differenzbildung

Gerade, wenn es darum geht, bewegende Objekte vor einem stillstehenden Hintergrund zu erkennen, kann noch vor dem Thresholding oder der Kantenerkennung eine Differenzbildung zwischen zwei (oder n) Bildern erfolgen.

Bei der Differenzbildung von zwei Bildern wird erst das erste Bild in den Speicher abgelegt und danach das nächste Bild in einen zweiten Speicherbereich. Wird nun jedes Pixel des ersten Bildes vom zweiten Bild abgezogen, bleiben nur die Pixel übrig, die sich zwischen den beiden Bildern geändert haben. Hierbei muss beachtet werden, dass Überschreitungen von Wertebereichen abgefangen werden müssen. Auch treten negative Differenzen zwischen zwei Bildern auf, so dass es sinnvoll ist, der Differenz der Bilder als Offset,

Segmentierung

50% des Wertebereichs der einem Pixel zu Verfügung steht, hinzu zu addieren. Ist das zweite Bild I_n , das erste Bild I_{n-1} und der größte Wert, den einem Pixel zu Verfügung steht 2^B (Oft $2^8=256$), so ergibt sich nach der folgend Formel (siehe Abb. Formel zu Differenzbildung mit Offset) ein Ausgangsbild I_{delta} .

Abbildung 3-24. Formel zu Differenzbildung mit Offset

$$I_n - I_{n-1} + 2^{B-1} = I_{\text{delta}}$$

In Bild I_{delta} würde die Position eines gleichmäßig hellen Rennwagens als helle Fläche dargestellt werden und die Position, an der sich der Rennwagen zum Zeitpunkt $n-1$ befand, ergäbe eine dunkle Stelle. Der Rest des Bildes wäre ein mittleres Grau. Bewegt sich der Rennwagen so langsam, dass es sich während zwei Bildern nur um ein Drittel der Fahrzeugfläche verschiebt, erkennt man das in Fahrtrichtung vordere Drittel als helle Fläche. Das Drittel in der Mitte ist nicht zu erkennen und das Drittel am Ende ergäbe eine dunkle Fläche. Da diese Technik nur zur Erkennung von Veränderungen im Bild geeignet ist, wird ein sich nicht bewegendes Rennwagen nicht erkannt.

Bei diesem Verfahren spielt die Güte einer Kamera eine große Rolle. Abhängig von der Technik und Qualität der Kamera sowie der Helligkeit, mit der ein Motiv ausgeleuchtet wird, enthalten die Bilder immer ein gewisses Rauschen, das sich zwischen zwei Bildern verändert, so dass das Ausgangsbild I_{delta} auch wieder ein Rauschen enthält. Ist dieses Rauschen zu hoch, führt es zu einer erheblichen Beeinflussung. Ein weiterer störender Effekt, auf den dieser Filter sehr empfindlich reagiert, ist der Schliereneffekt einer Kamera. Bewegt sich z.B. eine helle Fläche auf einem dunklen Hintergrund, so ziehen sich an der Kante, wo das helle Objekt über den dunklen Hintergrund wandert, dunkle Schlieren in das Objekt. An der Seite, wo das helle Objekt den dunklen Hintergrund wieder freigibt, ergeben sich helle Schlieren hinter dem Objekt. Ist dieser Effekt zu stark, wird die Kontur des Objektes aufgelöst, wodurch eine Erkennung erschwert wird.

Es muss bei dieser Technik ebenfalls beachtet werden, dass sie sehr empfindlich auf Helligkeitsschwankungen zwischen zwei Bildern reagiert. Dafür wird eine zeitlich gleichbleibende, ungleichmäßige Bild-Ausleuchtung kompensiert. Weiter reagiert dieser Filter sehr empfindlich auf Erschütterungen der Kamera durch die Produktion von starker Störinformation in kontrastreichen Gebieten des Bildes.

Beleuchtung

Besonders wichtig und kritisch bei der Segmentierung sind die Beleuchtungsumstände. Eine zu schwache Beleuchtung führt dazu, dass im Bild ein hohes Rauschen entsteht, was die Erkennung erschwert.

Die besten Ergebnisse erzielt man, wenn die Beleuchtungs- und die Betrachtungsachse möglichst parallel und nah beieinander liegen. Durch diese Ausleuchtung wird die Schattenbildung gering gehalten.

Wichtig ist, dass das Bild gleichmäßig ausgeleuchtet wird. So kann vermieden werden, dass sich die Farbe oder Helligkeit eines Objektes zu stark ändert, wenn es sich im Bild bewegt. Gerade durch die Verwendung von mehreren ähnlichen Lichtquellen, die parallel zur Betrachtungsachse das Bild ausleuchten, kann ein Kompromiss zwischen gleichmäßiger Ausleuchtung und geringerer Schattenbildung erzielt werden.

Als positiv hat sich die Verwendung von Objekte mit einer matten Oberfläche herausgestellt. Dadurch, dass

Segmentierung

das Licht durch die Oberfläche diffuser reflektiert wird, erscheint das Objekt in einer gleichmässigeren Farbe. Eine spiegelnde / glänzende Oberfläche bewirkt, dass Objekte und Lichtquellen in der Umgebung das Erscheinen des Objektes beeinflussen.

Zurück
Grabbing

Zum Anfang
Nach oben

Nach vorne
Flächenerkennung

Flächenerkennung

Wurde für ein Pixel das Thresholding abgeschlossen und es als Vordergrund-Pixel eingeordnet, kann in dem selben Schleifendurchlauf die Flächenerkennung für dieses Pixel erfolgen. Dadurch, dass die Flächenerkennung verschachtelt in das Thresholding geschieht, haben beide Abläufe die gleiche Verarbeitungsrichtung. Das heißt, dass Bild wird von links oben nach rechts unten zeilenweise abgearbeitet.

In der Methode `o_tracing::threshold` der Beispielapplikation, erkennt man, wie für jedes erkannte Vordergrund-Pixel noch aus der Thresholdfunktion heraus die Methode `o_tracing::labeling` aufgerufen wird. Hier wird deutlich, dass das Thresholding parallel zur Flächenerkennung erfolgt.

Die Aufgabe der Flächenerkennung ist es, ein Gebiet von Vordergrundpunkten, die aneinander angrenzen, als eine Fläche einzuordnen. Dabei gilt auch ein einzelner Punkt, der an keinen anderen angrenzt, als Fläche. Pixel, die aneinander angrenzen, werden in der Objekterkennung oft als Nachbarn bezeichnet.

Findung der Nachbarn

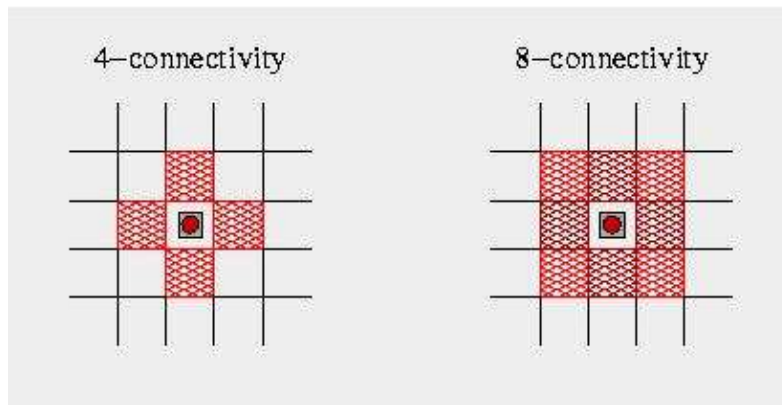
Die Erkennung von Nachbarn erfolgt nach folgendem Schema:

1. Es wird überprüft ob das aktuelle Pixel Nachbarn hat.
 - ◆ Hat das Pixel keinen Nachbarn, wird eine neue Gruppe erzeugt und dem Pixel zugewiesen.
 - ◆ Hat das Pixel einen oder mehrere Nachbarn der gleichen Gruppe, wird ihm die Gruppe seiner Nachbarn zugewiesen.
 - ◆ Hat das Pixel mehrere Nachbarn, die in verschiedenen Gruppen sind, muss eine Vereinigung der Gruppen vorgemerkt werden und das Pixel wird einer der Gruppen zugeordnet.
2. Der Vorgang wird mit dem nächsten Vordergrund-Pixel erneut durchlaufen bis alle Pixel abgearbeitet wurden.
3. In einem zweiten Durchlauf werden die zur Vereinigung vorgemerkten Gruppen zu einer Gruppe zusammengefasst.

Abhängig von der Definition der Nachbarschaft hat ein Pixel vier oder acht Nachbarn, durch die es beeinflusst wird. Bei vier Nachbarn der *4-connectivity* werden nur die unmittelbar anliegende Pixel als Nachbar gezählt, während es bei der *8-connectivity* acht Nachbarn gibt, da die diagonal anliegenden Pixel ebenfalls berücksichtigt werden. In Abb. 4- und 8-connectivity werden diese Verhältnisse dargestellt. Das zu beeinflussende Pixel ist in der Mitte durch einen roten Punkt markiert. Die Nachbar-Felder durch die es beeinflusst wird, sind rot schraffiert.

Abbildung 3-25. 4- und 8-connectivity

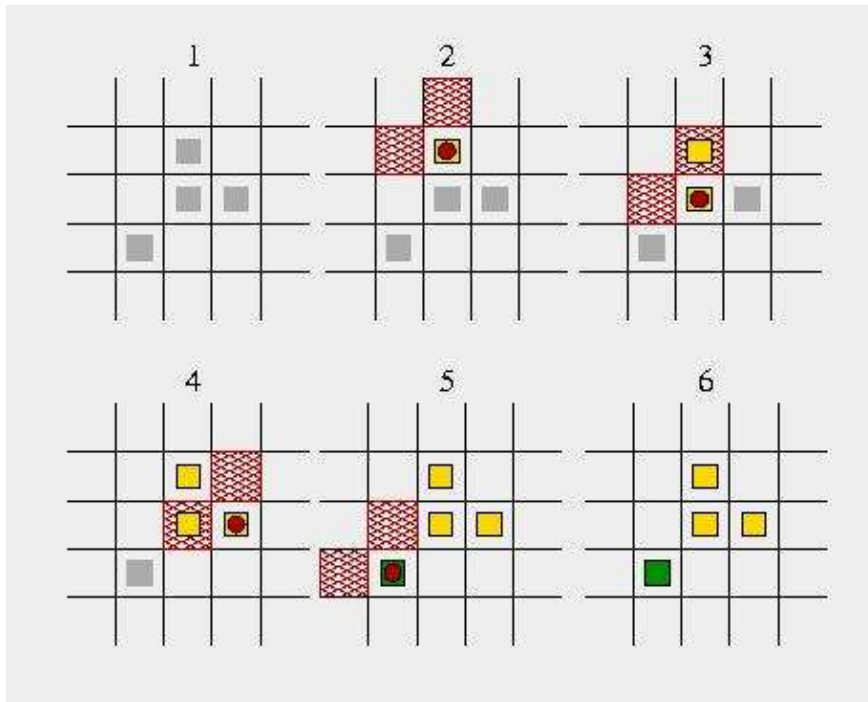
Flächenerkennung



Im Algorithmus wird bei der 4-connectivity nur das linke und obere Pixel nach einem Nachbarn abgetastet. Bei der 8-connectivity werden die Pixel links, links oben, oben und rechts oben untersucht. Zum einen würde zu diesem Zeitpunkt das Untersuchen der anderen anliegenden Pixel zu keinem Ergebnis führen, da diese noch nicht das Thresholding durchlaufen haben, zum anderen werden diese Punkte sowieso kurze Zeit später durch dem Algorithmus bearbeitet, welcher dann die Beziehung zu den zuvor erkannten Pixeln herstellt.

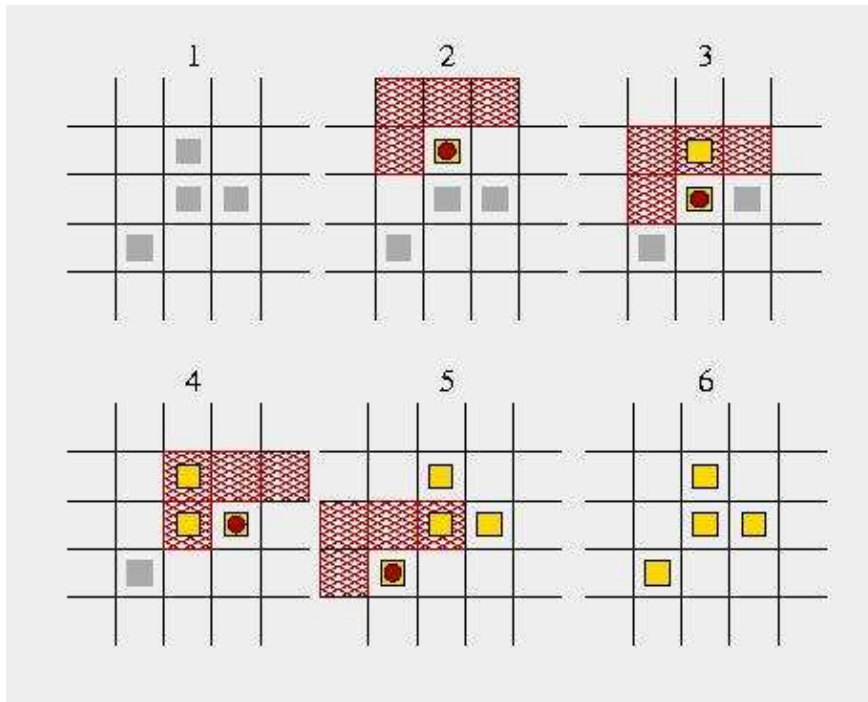
Betrachtet man in der Abb. 4-connectivity Nachbarerkennung die Teilbilder 1-6, erkennt man, wie die Pixel der Reihe nach abgearbeitet werden. In dem ersten Ausschnitt wurde noch kein Pixel bearbeitet oder erkannt. Die Tatsache, dass hier schon Vordergrund-Pixel grau hervorgehoben sind, dient nur zur Verdeutlichung ihrer Lage. Der Algorithmus kennt zu dieser Zeit noch keins der Pixel. Im zweiten Abschnitt wurde das erste Vordergrund-Pixel gefunden. Nun wird an den rot schraffierten Stellen geprüft, ob ein Nachbar vorhanden ist. Ist dies nicht der Fall, wird das Pixel einer neuen Gruppe zugewiesen, in diesem Fall durch die Farbe Gelb dargestellt. Bei Schritt 3 und 4 wird bei der Prüfung der Nachbarn erkannt, dass das Pixel an einem Nachbar angrenzt. Ihm wird die Gruppe seines Nachbars zugeteilt. Im 5. Teilbild besitzt das gerade untersuchte Pixel keinen direkten Nachbarn, deshalb wird ihm eine neue Gruppe zugewiesen. Diese Gruppe wird in der Abbildung durch die Farbe Grün dargestellt. Abschließend erkennt man in Teilabbildung 6, dass durch den Algorithmus 2 Flächen erkannt worden sind.

Abbildung 3-26. 4-connectivity Nachbarerkennung



In Abb. 8-connectivity Nachbarerkennung wird nochmals die gleiche Anordnung von Vordergrund-Pixeln bearbeitet. Allerdings wird diesmal die 8-connectivity verwendet, so dass jeweils vier Nachbar-Felder auf das Vorhandensein von Vordergrund-Pixeln überprüft werden. Diese vier Pixel sind z.B. in Teilabbildung 2 rot schraffiert dargestellt. Mit Ausnahme der erhöhten Anzahl von Nachbarn die überprüft werden, arbeitet der 8-connectivity Algorithmus genauso wie die 4-connectivity Technik. Sein Vorteil liegt darin, dass auch diagonal anliegende Nachbarn erkannt werden, so dass das Pixel in Teilbild 5, im Gegensatz zur 4-connectivity diesmal der gelben Fläche zugeordnet wird.

Abbildung 3-27. 8-connectivity Nachbarerkennung



Nach Tests mit beiden Verfahren wurde die Technik der 8-connectivity mit der Methode `o_tracing::labeling` implementiert. Die Betrachtung von nur vier Nachbarn neigt dazu, schmale diagonale Objekte in mehrere Einzelobjekte zu zerteilen. Insgesamt zeigte der 8-connectivity Algorithmus ein stärkeres Bestreben, Flächen zu bilden und trotz *Rauschen* beizubehalten als die 4-connectivity Variante. Da jedoch die 4-connectivity bis zu über 50 % weniger Rechenleistung benötigt und dennoch ausreichende Ergebnisse liefert, könnte zur Optimierung auf dieses Verfahren zurückgegriffen werden.

Randbehandlung

Beim Lesen der Nachbar-Pixel müssen drei Sonderfälle beachtet werden, da sonst falsche Informationen gelesen werden, und/oder Arraygrenzen verletzt werden.

1. Ist ein Pixel am linken Rand, dürfen keine Nachbarn zur Linken des Pixels gelesen werden.
2. Ist das Pixel am oberen Rand, dürfen keine Nachbarn oberhalb des Pixels gelesen werden.
3. Ist das Pixel am rechten Rand, dürfen keine Nachbarn rechts des Pixels gelesen werden.

Von diesen drei Bedingungen können auch mehrere gleichzeitig für ein Pixel zutreffen. So wird bei dem Pixel links oben in der Ecke eines Frames kein Nachbar überprüft, da sich alle Nachbar-Felder außerhalb des Frames befinden würden.

Eine entsprechende Implementierung findet man in der Beispielapplikation am Anfang der Methode `o_tracing::labeling`.

V-Problematik

Die V-Problematik beschreibt den Fall, dass die Nachbarn unterschiedlichen Gruppen angehören. Hat z.B. die zu erkennende Fläche das Aussehen des Buchstaben V, so wird die linke obere Ecke des Buchstaben der

Flächenerkennung

Gruppe 1 zugeordnet und die rechte obere Ecke der Gruppe 2. 1 und 2 sind Labels z.B. Integer Wert, die das Pixel der Gruppe 1 bzw. 2 zuordnen. Wenn beide Linien des Buchstaben unten zusammen laufen, stellt die Flächenerkennung fest, dass die beiden Gruppen 1 und 2 zu der selben Fläche gehören.

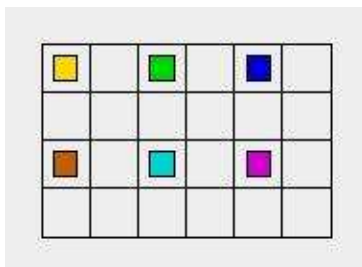
Eine Möglichkeit dieses Problem zu lösen, ist eine Tabelle anzulegen, in der jede Fläche einer andere Fläche zugeordnet werden kann. In dem Beispiel von oben würde $1=2$ in der ersten Zeile einer Tabelle eingetragen werden. Wie in Testbild zur V-Problematik verdeutlicht, kann es geschehen, dass sich mehrere Gruppen zu einer Fläche verbinden, so dass eine einfache 1 zu 1 Zuordnung nicht ausreicht, sondern eine 1 zu n Zuordnung benötigt wird. Praktisch müsste sich das Programm in der Tabelle merken, dass $1=2=3=4$ ist, um dann in einem zweiten Durchlauf alle Gruppen auf 1 zu setzen.

Abbildung 3-28. Testbild zur V-Problematik



In der Beispielapplikation heißt diese Tabelle LABELS und besteht unter anderem aus den zwei Integer Werten INDEX und NEXT_INDEX. Die Zeilenzahl der Tabelle ist $1/4$ der Pixelzahl, die ein Frame umfasst. Wenn alle Pixel bei einer 8-connectivity optimal vereinzelt sind, so dass jedes Pixel als ein eigenes Objekt erkannt wird, muss $3/4$ des Bildes mit Hintergrund gefüllt sein (siehe Abb. Größtmögliche Vereinzlung bei der 8-connectivity). Bei einer 4-connectivity müsste die Zeilenanzahl der Tabelle $1/2$ der Pixelzahl betragen, da sich die Pixel im Falle einer optimalen Vereinzlung wie die weißen Felder auf einem Schachbrett verteilen.

Abbildung 3-29. Größtmögliche Vereinzlung bei der 8-connectivity



Die Beispielapplikation implementiert eine Lösung für diese Problem zum Teil in der Methode `o_tracing::labeling`. Wird festgestellt, dass mehrere ungleiche Nachbarn vorhanden sind, werden

Flächenerkennung

diese in aufsteigender Reihenfolge sortiert. Nach der Sortierung werden Nachbarn die den Wert 0 enthalten ignoriert, da es sich hierbei um Hintergrund-Pixel handelt. Durch die Sortierung stehen nun gleiche Gruppen nebeneinander, so dass durch paarweises Vergleichen schnell herausgefunden werden kann, welche Nachbarn ungleich sind. Für jedes ungleiche Nachbarpaar wird jetzt die Methode `o_tracing::in_label` aufgerufen, um die zwei Label in eine Tabelle einzufügen.

Die Methode `o_tracing::in_label` bekommt als Parameter die beiden Label. Nun wird bei dem vom Wert kleineren Label das andere Label in die Variable `NEXT_INDEX` eingetragen. Für den Fall, dass `NEXT_INDEX` schon einen Wert enthält, wird der Wert in `NEXT_INDEX` und der Wert, der eben nicht eingetragen werden konnte, als Parameter für einen neuen Aufruf der Methode `o_tracing.in_label` verwendet. Dieser rekursive Ablauf geschieht so lange bis der neue Index eingetragen wurde. Hierbei wird und muss sichergestellt werden, dass nie der `NEXT_INDEX` Eintrag kleiner als der `INDEX` Eintrag ist, da sich sonst Zirkularitäten ergeben, die ein schnelles Auslesen der Daten erschweren. Für das Testbild zur V-Problematik ergibt sich durch den Algorithmus folgende **LABELS Tabelle**. Auf die Spalten `MENGE`, `X_MIN`, `X_MAX`, `Y_MIN`, `Y_MAX` wird in Kapitel Objektbewertung eingegangen. Die Spalte `ROOT` wird im Folgenden erläutert.

Abbildung 3-30. LABELS Tabelle

INDEX	NEXT_I.	MENGE	X_MAY	X_MIN	Y_MAX	Y_MIN	ROOT
0	0	0	0	0	0	0	0
1	2	8	71	62	28	13	1
2	4	9	67	55	39	19	1
3	4	6	61	28	52	28	1
4	0	21	60	38	70	30	1

Wurden alle Pixel mit einem Label markiert, kann in einem zweiten Durchlauf damit begonnen werden, die Gruppen / Labels zu vereinigen. Das würde jedoch bedeuten, dass in der Tabelle LABELS für jedes Pixel überprüft werden muss, ob der Punkt nicht mit einer anderen Gruppe zu vereinigen ist. Um nicht bei jedem Pixel durch die LABELS Tabelle traversieren zu müssen, wird diese Tabelle einmalig komplett durchlaufen und die Spalte `ROOT` gebildet. `ROOT` gibt je nach Implementierung das größte oder kleinste Label einer Fläche an. Eine Implementierung, die eine solche Aufgabe übernimmt, findet man in der Methode `o_tracing::create_roots`, die zu Beginn der Methode `o_tracing::join_labels` für alle Zeilen der Tabelle LABELS aufgerufen wird.

Wird jetzt das Bild zum zweitenmal durchlaufen, findet nur ein Tabellen Zugriff für jedes Pixel statt. Mit dem LABEL-Wert des Pixels wird in der entsprechenden LABELS-Tabellenzeile der `ROOT`-Wert ausgelesen und an die Pixelposition geschrieben. Am Ende des Durchlaufes verfügen alle Flächen über das selbe Label. Nun kann begonnen werden, die Flächen nach bestimmten Kriterien zu untersuchen, um ein bestimmtes Objekt zu erkennen.

[Zurück](#)
Segmentierung

[Zum Anfang](#)
[Nach oben](#)

[Nach vorne](#)
Objektbewertung

Objektbewertung

Wurden alle Flächen durch die Flächenerkennung erkannt, kann begonnen werden nach der Fläche zu suchen, die das zu verfolgende Objekt repräsentiert.

Dabei ist wichtig, dass weitere Informationen über die Flächen in einer Tabelle gesammelt werden. Da durch das Thresholding bereits eine Selektion z.B. nach Farben durchgeführt wurde, bleiben nun Kriterien wie die Menge der Pixel, die die Fläche umfasst, die Pixeldichte (Pixelzahl pro Fläche), die Position im Bild oder eine markante Form als Selektionsattribute übrig.

Anstatt nun erneut jede Fläche zu analysieren, erfolgt diese Aufgabe schon während der Flächenerkennung. So wird in der Beispiellapplikation bei jedem erkannten Vordergrund-Pixel die Methode `tracing::just_count` aufgerufen, um Informationen über die Fläche zu sammeln. Da sich diese Informationen immer auf die entsprechende Fläche beziehen, werden sie in der Tabelle LABELS abgelegt und einem Label / einer Gruppe zugeordnet. Werden im zweiten Durchlauf Gruppen zu einer Fläche vereinigt, so werden auch die gesammelten Informationen zusammengeführt.

Mengendichte

In der Beispiellapplikation werden unter anderem als zusätzliche Informationen die Pixel-Menge und die Ausdehnung der Fläche abgelegt.

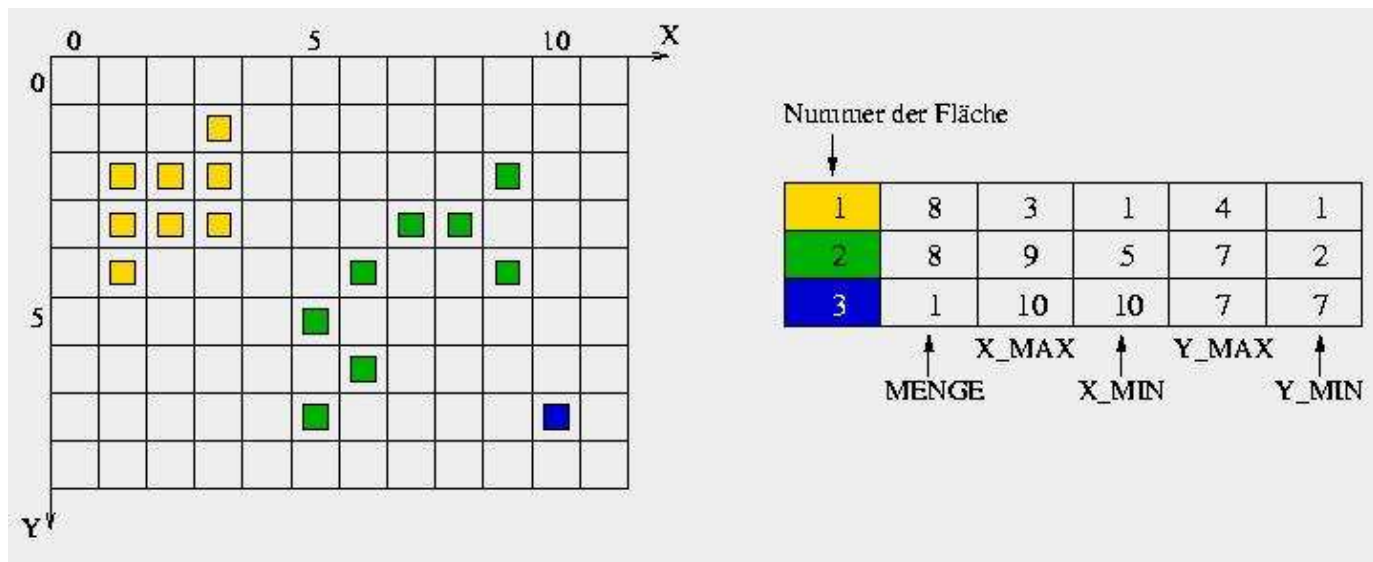
Bei der MENGE handelt es sich um die Anzahl der Pixel, die eine Fläche umfasst. Sie wird für jedes Pixel, das einer Fläche zugeordnet wird, um eins inkrementiert.

Die Ausdehnung und Position der Fläche im X-Y-Koordinatensystem des Bildes wird in den Spalten X_MIN, X_MAX und Y_MIN, Y_MAX abgelegt. Hierbei wird für jedes neue Pixel dieser Gruppe überprüft, ob dies die bestehenden maximalen Ausdehnungen der Fläche erweitert. Das bedeutet, dass der Wert für ein Pixel ersetzt wird, wenn der X-Wert des aktuellen Pixels kleiner ist als der bis jetzt gespeicherte X_MIN Wert. Ist er größer, so wird der bis dahin ermittelte X_MIN-Wert beibehalten.

Das X-Y-Koordinatensystem, welches dabei verwendet wird, ist wie in Kapitel [Segmentierung](#) Abschnitt [Verarbeitungsrichtung](#) schon erläutert, kein vorliegendes System. So wird bei einer Division durch die Bildbreite die Zeilennummer durch den ganzzahligen Anteil repräsentiert. Die Position in der Zeile ergibt sich aus dem Rest des Quotienten.

In Abb. [Objekte zur Auswahl](#) sind drei Flächen dargestellt, wovon das gelbe Objekt links oben der Rennwagenkandidat ist. Daneben befindet sich eine Aufstellung, der während einer Flächenerkennung gesammelten Informationen, wie sie zum Teil in der Tabelle LABELS stehen würden.

Abbildung 3-31. Objekte zur Auswahl



Wie man in Zeile 1 und 2 erkennt, reicht es nicht aus das Objekt anhand der MENGE zu ermitteln. Da es sich bei dem Rennwagen um einen kompakten Körper handelt, ist ein weiteres betrachtenswertes Kriterium die Dichte des Körpers. Spricht man hier von Dichte, ist damit das Verhältnis zwischen vom Objekt belegter Vordergrund-Pixel zur vom Objekt beanspruchten Fläche gemeint.

In der Beispielapplikation ist die Fläche eine Multiplikation der Breite, die durch $X_MAX - X_MIN + 1$ angegeben wird, mit der Höhe, die sich durch $Y_MAX - Y_MIN + 1$ ergibt. Die Notwendigkeit der "+1" erkennt man bei der Betrachtung einer Fläche, die nur aus einem Pixel besteht. Bei dieser ist der X_MIN -Wert gleich dem X_MAX -Wert. Eine Differenz der beiden Werte ergibt 0, aber der Körper hat die Breite 1. Die Formel zur Errechnung der Dichte gibt diese Zusammenhänge wieder.

Abbildung 3-32. Formel zur Errechnung der Dichte

$$\text{Dichte} = \frac{\text{MENGE}}{(X_MAX - X_MIN + 1) * (Y_MAX - Y_MIN + 1)}$$

Wendet man diese Formel auf die Abb. Objekte zur Auswahl an, so ergeben sich folgende Ergebnisse:

$$\text{Dichte}_1 = 0,67$$

$$\text{Dichte}_2 = 0,27$$

$$\text{Dichte}_3 = 1,00$$

Man erkennt, dass Fläche 3 zwar die höchste Dichte hat, aber aufgrund ihrer geringen Pixelmenge einen schlechten Rennwagenkandidat darstellt.

Objektbewertung

Um die MENGE stärker in die Gewichtung einfließen zu lassen, wird die Dichte mit der MENGE multipliziert. Das Ergebnis dieser Berechnung erhält den Namen Mengendichte. Verdeutlicht in der Formel zur Errechnung der Mengendichte.

Abbildung 3-33. Formel zur Errechnung der Mengendichte

$$\text{Mengendichte} = \frac{\text{MENGE}^2}{(\text{X_MAX} - \text{X_MIN} + 1) * (\text{Y_MAX} - \text{Y_MIN} + 1)}$$

Mengendichte₁=5,33

Mengendichte₂=2.13

Mengendichte₃=1,00

Betrachtet man die Ergebnisse, repräsentiert nun die höchste Mengendichte das potentielle Rennwagenobjekt.

Berechnet man nach dieser Formel z.B. die Mengendichte einer Fläche von 924*576 Pixel, so würde sich für das Teilergebnis MENGE² ein Wert von 283.262.386.200 ergeben. Für diesen Wert wäre z.B. eine 32 Bit umfassende Variable zu klein (2³²=4.294.967.296). Um eine Bereichsüberschreitung durch solche hohen Werte zu vermeiden, wird die Gleichung in einer bestimmten Reihenfolge ausgeführt. Des Weiteren muss untersucht werden, welche Werte durch welche Berechnung im schlimmsten Fall erreicht werden können.

tmp1 = (X_MAX-X_MIN+1)*(X_MAX-X_MIN+1);

Das Ergebnis für tmp1 ist maximal gleich der Anzahl aller Pixel im Frame. Bei der maximalen Auflösung, der für den Versuch verwendete Hauptauge PCI Frame-Grabber-Karte sind das 924*576=532.224 Pixel. Das entspricht dem Wert, der für MENGE erreicht werden kann.

Um das rechenleistungsintensive Rechnen mit Nachkommastellen zu vermeiden, wird die MENGE oberhalb des Bruchstrichs mit 100 multipliziert. Die MENGE*100 wird so zu maximal 53.222.400.

tmp2 = (MENGE*100) / tmp1;

Für tmp2 ist der kleinste mögliche Wert, die Breite oder die Höhe des Bildes, je nachdem was den größeren Wert darstellt, geteilt durch die Anzahl aller Pixel im Frame. Ein solches Bild wäre eine diagonale Linie vom obersten linken zum untersten rechten Punkt des Frames. Für eine Auflösung von 924*576 Bildpunkten ergibt sich für tmp2 so ein Minimumwert von 924*100/532.224 = 17,36. Den minimalen Wert 1,00 für tmp2 erhält man bei einer quadratischen Fläche (Höhe=Breite) durch die eine Diagonale verläuft. Der maximale Wert 100 ergibt sich bei einer ausgefüllten Fläche.

Durch die in C übliche Division werden Nachkommastellen gestrichen, so dass 17,36 zu 17 wird. Das Ergebnis liegt also innerhalb der Grenzwerten 1 und 100. Dieser Wert entspricht der oben genannten Dichte und gibt das Verhältnis gesetzter Pixel zu nicht gesetzter Pixel in Prozent an.

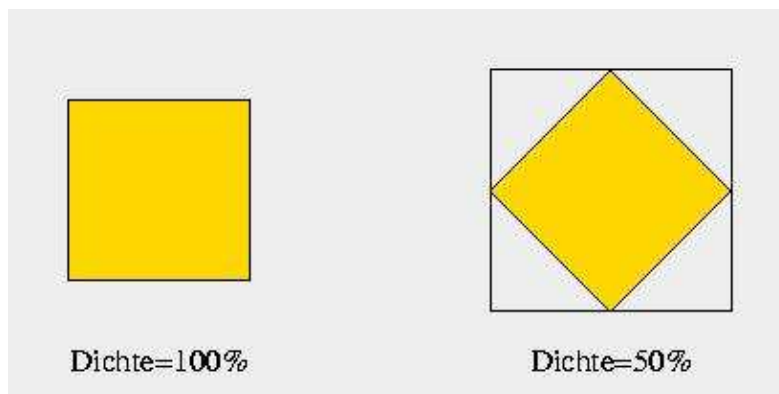
Mengendichte=MENGE*tmp2;

Da dieses Ergebnis mit MENGE multipliziert wird, erhält man bei dieser Auflösung einen Maximalwert von 53.222.400, welcher leicht in eine 32 Bit großen Variable aufgenommen werden kann.

Die Ermittlung der Mengendichte wird anschließend an die Flächenerkennung in der Methode `tracing::join_labels` durchgeführt. Dort wird auch ermittelt welches die Fläche mit der größten Flächendichte ist, um sie als zu verfolgendes Objekt abzulegen. Die eigentliche Berechnung übernimmt die Methode `tracing::get_mengendichte`.

Ein Schwachpunkt dieser einfachen Ermittlung der Mengendichte tritt bei bestimmten sich drehenden Objekten auf. Dadurch, dass die Fläche, die ein Objekt einnimmt, lediglich durch die Betrachtung der Minimum und Maximum-Werte in X und Y-Richtung erfolgt, liegt ihr immer eine quadratische Form zu Grunde. Analysiert man z.B. ein Quadrat, das horizontal ausgerichtet ist, so erhält es eine Dichte von 100%. Dreht sich nun das Quadrat um 45 Grad nach rechts oder links, sinkt die Dichte auf 50%. Verdeutlicht wird das in Abb. Änderung der Dichte bei Drehung.

Abbildung 3-34. Änderung der Dichte bei Drehung



Eine Runde Fläche hingegen wird durch eine Drehung nicht beeinflusst, wodurch die Dichte gleich bleibt.

Kreisflächenähnlichkeit

Bei der Suche nach einer Technik, die eine geringere Empfindlichkeit gegenüber der Rotation der Objekte aufweist als die Ermittlung der Flächendichte und zusätzlich eine Aussage über die Dichte einer Fläche ergibt, wurde in dieser Arbeit ein Verfahren zur Ermittlung der Kreisflächenähnlichkeit entwickelt.

Grundlage des Verfahren ist es, die Anzahl der CONNECTIONS einer Fläche in ein Verhältnis zu der Menge der CONNECTIONS einer errechneten Kreisfläche mit gleicher Pixelzahl zu bringen. Mit Anzahl oder Menge der CONNECTIONS ist hier die Summe der Verbindungen gemeint, die zwischen den Pixeln einer Fläche existieren. Da eine Kreisfläche gegenüber gleich großen, anders geformten Flächen die höchste Dichte hat, kann mit dieser Technik auf die Dichte einer Fläche geschlossen werden.

Für diese Verfahren wird für alle Pixel einer Fläche, die Summe der erkannten Nachbarn ermittelt. Dies lässt sich schon während der Flächenerkennung durchführen. In der Beispielapplikation wird aus der Methode

Objektbewertung

`o_tracing::labeling` die Methode `o_tracing::just_count` aufgerufen. Sie erhält als Parameter die Anzahl der gefundenen Nachbarn und addiert sie, in der Spalte CONNECTIONS der Tabelle LABELS, zu der jeweiligen Gruppe. Werden später zwei Gruppen vereinigt, erfolgt eine Aufsummierung der CONNECTIONS-Werte.

Der CONNECTIONS-Wert, der sich so für eine Fläche ergibt, kann nun in ein Verhältnis zu einem errechneten CONNECTIONS-Wert für die Fläche eines Kreises gebracht werden. Dabei ist die Menge der Pixel des Objektes gleich der Menge der Pixel der errechneten Kreisfläche.

Um für eine solche Kreisfläche, unter Angabe einer Pixel-Menge, einen CONNECTIONS-Wert zu errechnen, wird die Pixelmenge mit der Anzahl der 1/2 Connectivity, die z.B. in der Beispielapplikation $1/2 * 8 = 4$ beträgt, multipliziert. Von diesem Ergebnis wird die Anzahl der Randpixel, welche mit 1/4 der Connectivity multipliziert wurden, abgezogen. Die ist nötig, da die Randpixel eines Kreises eine um 50% geringere Anzahl von CONNECTIONS aufweisen. Zur Errechnung der Anzahl der Randpixel wird die Kreisfläche mit Hilfe der Pixelanzahl errechnet und von einer Kreisfläche, welche einen um ein Pixel kleineren Radius aufweist, abgezogen.

Nach einer Vereinfachung ergibt sich die Formel der Kreisflächenähnlichkeit. In der Formel wird die Menge durch den Buchstaben M dargestellt.

Abbildung 3-35. Formel der Kreisflächenähnlichkeit

$$\text{Kreisflächenähnlichkeit} = \frac{\text{CONNECTIONS} * 4}{\text{Connectivity} * (-2 * \sqrt{M * \pi} + 2 * M + \pi)}$$

Für Formen, die einer Kreisfläche entsprechen, ergibt diese Berechnung einen Wert um 1. Je weiter sich ein Körper von der optimalen Dichte einer Kreisfläche entfernt, umso kleiner wird das Ergebnis. Ein Sonderfall ist eine Fläche, die nur einen Punkt umfasst. Als Ergebnis wird hier der Wert 0 errechnet.

Da sich in der Bildverarbeitung eine Kreisfläche nicht ohne Kompromisse auf einen Pixelraum abbilden lässt, ist bei einem Vergleich zwischen einer Abbildung in einem Pixelraum und einem errechneten Wert immer mit einem Fehler zu rechnen. Je höher die Auflösung des Pixelraumes und die Größe eines Objektes ist, desto geringer wird dieser Fehler.

Vereinfacht kann auch das Verhältnis CONNECTIONS/MENGE als ein Maß der Dichte genommen werden. Für sehr große Kreisflächen läuft dieses Ergebnis gegen den Wert der halben Connectivity. Dieses Verhalten begründet sich dadurch, dass die Beeinflussung durch den Rand gegenüber der Gesamtfläche, bei steigender Pixelzahl, immer geringer wird, da sich die Fläche gegenüber dem Umfang eines Kreises quadratisch erhöht $A = U^2 / (4 \pi)$.

Genau wie bei der Mengendichte kann durch Multiplikation mit der Menge die Pixelzahl in das Ergebnis miteinbezogen werden. Da $\text{CONNECTIONS/MENGE} * \text{MENGE} = \text{CONNECTIONS}$ ist kann der CONNECTIONS-Wert direkt als Kriterium für große Flächen mit hoher Dichte verwendet werden.

Der Vorteil der Bewertung durch den CONNECTIONS-Wert liegt darin, dass der verwendete Algorithmus eine hohe Verarbeitungsgeschwindigkeit aufweist.

Objektbewertung

Eine Objektbewertung wie sie durch die Kreisflächenähnlichkeit (und die Mengendichte) in der Beispielapplikation beschrieben wird, kann nur zur Differenzierung einfach geformter Körper, die sich von ihrem Hintergrund abheben, verwendet werden. Da die Farbe und die Kreisflächenähnlichkeit als Attribut für die Objektidentifikation dienen, ist eine Unterscheidung zwischen ähnlichen gleichfarbigen Objekten wie beispielsweise einem Achteck und Neuneck nicht möglich.

Betrachtet man in der Praxis zwei Modell-Rennwagen auf einer Rennstrecke, sind beide Objekte sehr ähnlich geformt. Spätestens bei der Abbildung im Speicher auf 15 Pixel je Fahrzeug wird eine Unterscheidung anhand der Form sehr schwierig. Erheblich effizienter ist es, diese Pixelanhäufungen des Rennwagens anhand seiner Farbe und Kompaktheit bzw. Kreisflächenähnlichkeit zu unterscheiden.

Die Selektion

Im einfachsten Fall wird jetzt das Objekt gewählt, welches das Kriterium am Besten erfüllt. Bei der Mengendichte oder dem CONNECTIONS-Wert wäre dies das Objekt mit dem höchsten Wert.

Je nach Anwendung und zu erkennendem Objekt ist es sinnvoll, nicht einfach den Maximalwert zu wählen, sondern einen Wert, der sich nahe eines bekannten Wertes befindet. Ein Beispiel dafür wäre, dass die Größe des zu findenden Objektes bekannt ist. Ermöglicht es die Aufgabenstellung, ist es empfehlenswert, Wertebereiche zu definieren, in denen sich das Kriterium befindet oder umgekehrt gewisse Wertebereiche auszuschließen. So lässt sich z.B. vermeiden, dass ein im Rennbahnbereich liegendes Blatt Papier als Rennwagen erkannt wird oder fälschlicherweise beim Fehlen eines Rennwagen ein Fussel auf der Fahrbahn verfolgt wird.

Sollen in einem Bild mehrere Objekte verfolgt werden, ist es je nach Implementierung nötig, das erkannte Objekt als schon erkannt zu markieren oder es auf dem Bild nach seiner ersten Erkennung zu löschen. So wird vorgebeugt, dass das selbe Objekt mehrfach erkannt wird. Diese Technik schützt jedoch nicht davor, dass zwei Objekte bei der Erkennung vertauscht werden. So kann es geschehen, dass nach einer Fahrbahnkreuzung (Schikane) zwei Rennwagen vertauscht wiedererkannt werden. Um dieses zu vermeiden, müssen die Fahrzeuge schon zuvor durch unterschiedliche Kriterien voneinander getrennt worden sein oder es muss sichergestellt werden, dass sie nie in den selben Objekterkennungsbereich geraten.

(Die Technik zur Verfolgung eines Objektes und der Wahl des Objekterkennungsbereiches wird im nächsten Kapitel beschrieben.)

Eine Gefahr der Selektion durch die Mengendichte oder den CONNECTIONS-Wert ist, dass es vorkommen kann, dass das verfolgte Objekt eine andere gleichfarbige Fläche tangiert und beide zu einer neuen Fläche vereinigt werden. Hierdurch ändern sich die Eigenschaften des verfolgten Objektes, was zu einer Nichterkennung oder Fehlererkennung führen kann. Bei der Modell-Rennbahn lässt sich dies dadurch vermeiden, dass für die Rennwagen zwei unterschiedliche Farben verwendet werden, die in der unmittelbaren Umgebung der Rennstrecke nicht als größere Fläche auftauchen.

[Zurück](#)
Flächenerkennung

[Zum Anfang](#)
[Nach oben](#)

[Nach vorne](#)
Subframing / Verfolgung

Subframing / Verfolgung

Unter Subframing versteht man, dass nicht das gesamte Bild nach dem Objekt durchsucht wird, sondern nur der Teilausschnitt, in welchem das Objekt erwartet wird.

Diese Technik hat den Vorteil, dass für die Vorgänge des Thresholding und Labeling nur ein kleiner Bereich bearbeitet werden muss. Dies wirkt sich auch auf die Anzahl der von der Selektion zu bearbeitender Objekte aus. So ist es möglich, in jedem dieser Bereiche Rechenleistung einzusparen und dem Wunsch nach Einhaltung der 1. und 2. Echtzeitbedingung entgegenzukommen.

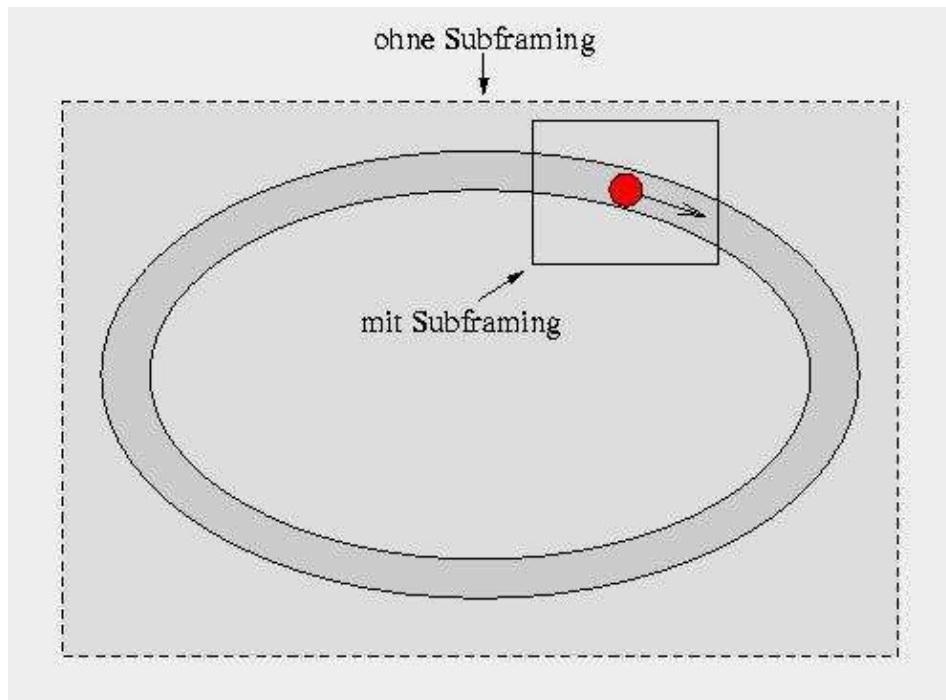
Je nach Frame-Grabber-Karte wird sogar das Grabben eines Subframes aus dem Bildstrom unterstützt. Diese Technik hätte zwar den Vorteil, dass sich das Datenaufkommen schon beim Grabvorgang reduzieren liesse. Allerdings würden bei der Verfolgung mehrerer Objekte mehrere solcher Subgrabvorgänge anfallen, wodurch wieder mehr Zeit in Anspruch genommen werden müsste. Zum anderen hätte die Applikation mit dynamischen Speicherbereichen zu arbeiten, was sich ebenfalls negativ auf die Bearbeitungszeit des Prozesses auswirken würde.

Ein anderer Vorteil des Subframing ist, dass nur der Bereich um das Objekt durchsucht wird, so dass Störungen außerhalb des Bereiches nicht wahrgenommen werden.

Nachteilhaft bei dieser Technik ist, dass ein Objekt, das nicht an der erwarteten Position, sondern außerhalb des Subframe erscheint, nicht erkannt wird. Außerdem beansprucht die Berechnung der Position Rechenleistung. Diese ist zwar im Verhältnis zur Einsparung durch ein gutes Subframing sehr gering, es kann jedoch vorkommen, dass durch eine Wahl eines sehr großen Subframe die Gesamtrechenleistung höher wird als bei einem Verfahren ohne Subframing.

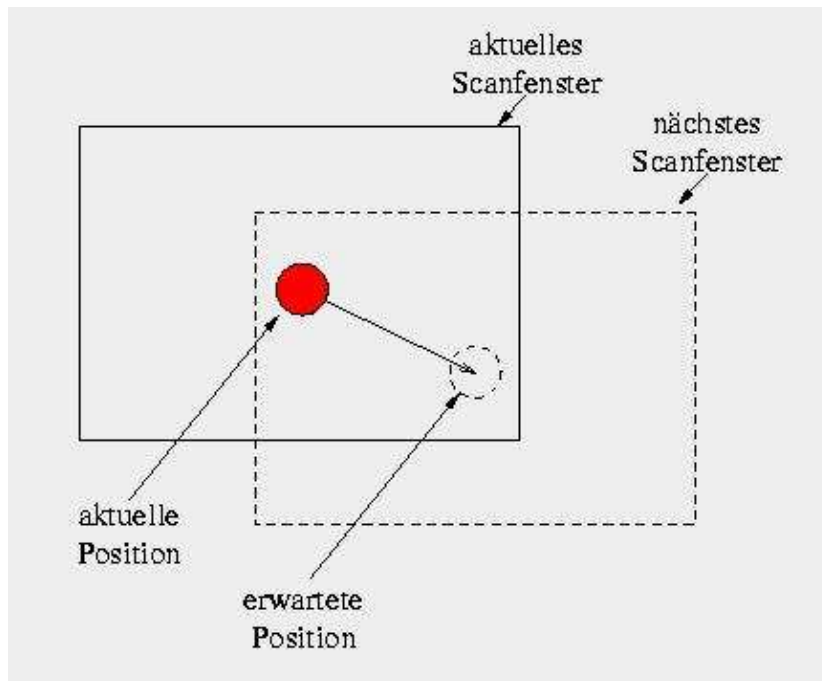
In Abb. Bearbeitungsbereiche beim Subframing ist das zu verfolgende Objekt durch einen roten Punkt dargestellt, der sich auf der ovalen dunklen Bahn in Pfeilrichtung bewegt. Man erkennt deutlich, dass die Fläche des Subframe, die als Kästchen um den Punkt dargestellt ist, erheblich kleiner ist als die Fläche des gesamten Frames, der in der Abb. Bearbeitungsbereiche beim Subframing durch die gestrichelte Linie eingeschlossen wird.

Abbildung 3-36. Bearbeitungsbereiche beim Subframing



Möchte man beim Subframing ein Objekt verfolgen, so ist es nötig, die Position des Objektes zu extrapolieren. In Abb. Platzierung des Subframe ist ein aktueller Subframe und der Subframe in dem das Objekt erwartet wird, dargestellt.

Abbildung 3-37. Platzierung des Subframe



Die Position, an der das Objekt erwartet wird, kann wie in der Methode `o_tracing::tracing` der Beispielapplikation errechnet werden. In dieser Methode wird mit Hilfe der in der Labels-Tabelle abgelegten

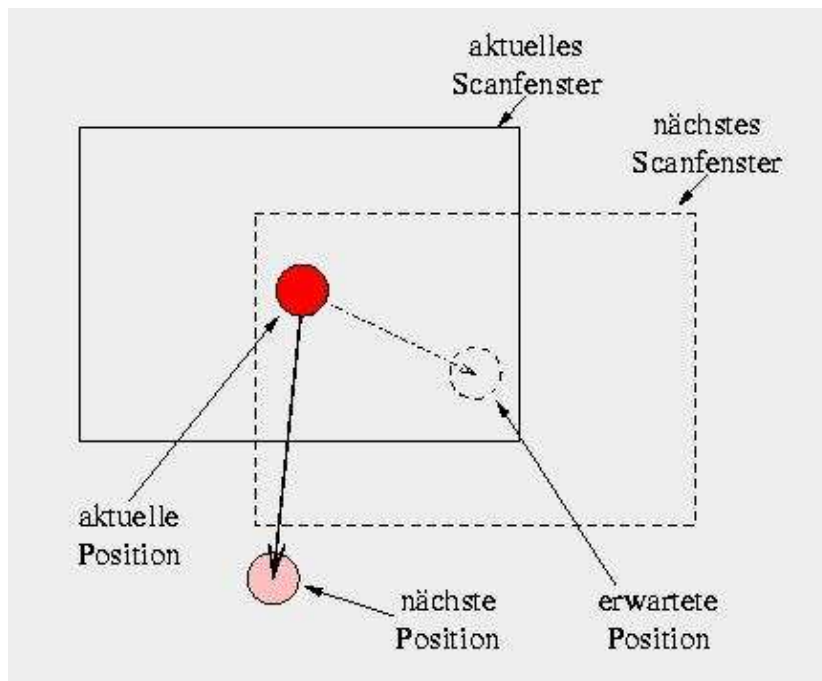
min. und max. Koordinaten, die Mitte eines um den Körper ausgespannten Viereckes errechnet. Addiert man diese zu den min. Koordinaten des Objektes, erhält man die Lage des Objektes im Frame. Zieht man von diesem Punkt die Koordinaten des im vorhergehenden Frame genauso errechneten Punktes ab, so ergibt sich die Richtung, in die sich das Objekt seit dem letzten Frame bewegt hat. Addiert man diese Richtung zur aktuellen Position des Objektes, ergibt sich der Punkt, an dem das Objekt erscheinen wird, wenn es seine Richtung und Geschwindigkeit beibehält.

Der Subframe sollte ein Quadrat um diesen Punkt darstellen, das mindestens um ein Pixel auf beiden Achsen größer ist als die viereckige Fläche, in der sich das Objekt befindet. Würde die Subframegröße genauso groß oder kleiner als die Objektgröße gewählt werden, hätte das Objekt keine Möglichkeit größere Ausmaße anzunehmen, wie es beispielsweise durch eine Drehung eines nicht kreisrunden Körpers geschieht. Da das Fenster jedoch bei jeder Möglichkeit kleiner werden würde, ergäbe sich nach kurzer Zeit eine Subframegröße von null. Um dieses zu vermeiden und um sicherzustellen, dass das Objekt auch noch erkannt wird, wenn es nicht genau auf der errechneten Position erscheint, wird z.B. in der Beispielapplikation die Fenstergröße doppelt so groß wie die Objektbreite gewählt. Darüber hinaus wird das Subfenster noch durch einen dynamischen Faktor in die Bewegungsrichtung vergrößert, der von der Geschwindigkeit des Objektes abhängt.

Sobald sich ein Objekt dem Rand des Frames (nicht Subframe) nähert, besteht die Gefahr, dass der Subframe die Grenzen des Frame Array oder die logische Höhen- Breitereinteilung verletzt. Deshalb müssen diese Grenzen überwacht und gegebenenfalls der Subframe eingeschränkt werden.

Bewegt sich das Objekt zu unregelmäßig oder wurde der Subframe zu klein gewählt, kann es geschehen, dass das Objekt verloren geht (siehe Abb. Verlust des Objektes). In einem solchen Fall muss der Subframe wieder vergrößert werden, um das Objekt wieder zu finden.

Abbildung 3-38. Verlust des Objektes



Wurde das Objekt erfolgreich erkannt, kann begonnen werden, diese Informationen weiterzugeben. In der

Subframing / Verfolgung

Beispielapplikation geschieht dies durch die Wiedergabe des durch die Kamera aufgenommenen Bildes in einem Fenster. Ein verfolgtes Objekt wird hierbei durch kleine Ecken an den Kanten des Subframefensters hervorgehoben. Die entsprechende Implementierung befindet sich in der Methode `o_tracing::draw_tracing_frame`.

[Zurück](#)

Objektbewertung

[Zum Anfang](#)

[Nach oben](#)

[Nach vorne](#)

Grafisches-Fenster

Kapitel 4. Grafisches-Fenster

Inhaltsverzeichnis

[X-Window-Client-Server-Prinzip](#)

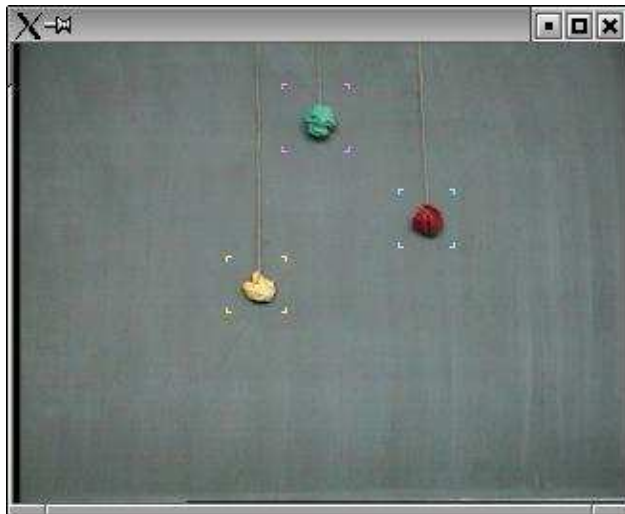
[X-Window-Nachrichten](#)

[Xlib-Fenster](#)

Während sich die vorhergehenden Kapitel vorwiegend mit der Eingabe und Verarbeitung von Bildinformationen befasst haben, beschreibt dieses Kapitel die anschliessende Ausgabe der erhaltenen Information in einem Fenster, sowie die Verarbeitung von interaktiven Benutzereingaben.

Gerade bei der Objekterkennung und Objektverfolgung ist es oft sinnvoll zu sehen wie eine Applikation arbeitet. Das ist auch der Grund, weshalb die Beispielapplikation über ein grafisches Fenster verfügt. Zu einem früheren Entwicklungszeitpunkt liessen sich mit diesen Methoden beispielsweise alle erkannten Flächen in verschiedenen Graustufen darstellen. So war es möglich, die korrekte Arbeit der Segmentierung zu überprüfen. In der dieser Arbeit beiliegenden Version wird das von der Kamera aufgenommene Bild angezeigt. Um die verfolgten Objekte hervorzuheben, wird der jeweilige Subframe um ein Objekt durch Kantenmarkierungen hervorgehoben (siehe Abb. *drei Papierkugeln in Subframes*).

Abbildung 4-1. drei Papierkugeln in Subframes

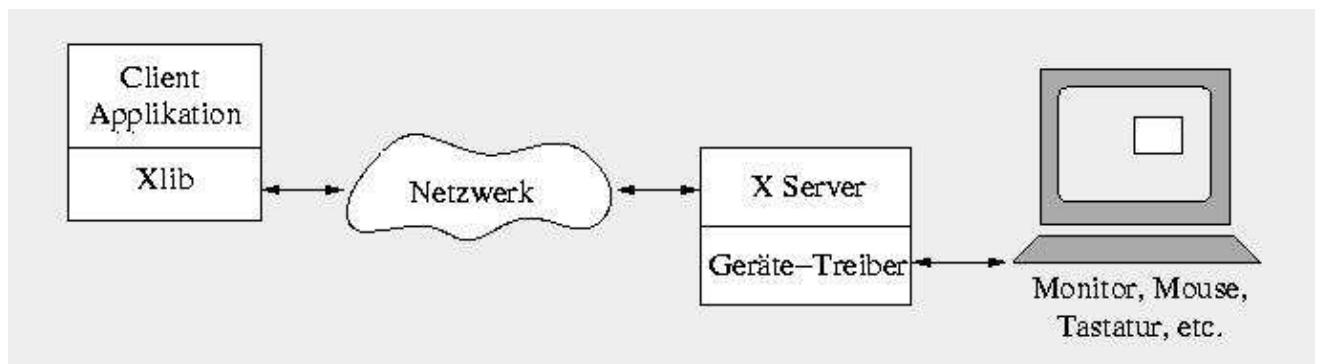


Für die Implementierung des Grafischen-Fensters wurde die C Library Xlib benutzt. Sie ermöglicht einen rudimentären Zugriff auf das X-Window-System. Bevor jedoch weiter auf die Verwendung der Xlib eingegangen wird, ist es sinnvoll, den groben Aufbau des X-Window-System zu betrachten.

X-Window-Client-Server-Prinzip

Wie in Abb. *Das X-Window-Client-Server-System* dargestellt, wird das X-Window System in eine Client-Seite und eine Server-Seite unterteilt. Auf der Server-Seite läuft der X-Server, eine Applikation, die den Zugriff auf die Eingabe und Ausgabe Schnittstellen wie Tastatur, Maus, Monitor oder ähnlichem steuert. Auf der Client-Seite werden eine oder mehrere Applikation betrieben, welche über Nachrichten mit dem Server kommunizieren können. Ein Beispiel für eine solche Applikation ist das Beispielpogramm dieser Arbeit. Diese Unterteilung in Client und Server hat die Vorteile, dass eine Applikation auf einem Rechner ablaufen kann, während sie durch einen anderen Rechner über ein Netzwerk gesteuert wird. Natürlich können sich Client und Server auch auf dem gleichen Rechner befinden. Ein weiterer Vorteil dieser Trennung ist, dass beide Programmteile nicht in der gleichen Programmiersprache geschrieben worden sein müssen. Es reicht aus, wenn sie über das einheitliche X-Protokoll kommunizieren. Bei der Zuordnung der Begriffe Client und Server fällt auf, dass diese Einteilung nicht der in der Netzwerntechnik üblichen Definition entspricht. In Computer Netzen wird mit Server ein Rechner bezeichnet, der Informationen oder Rechenleistung für Clients bereitstellt. Der Anwender befindet sich in der Regel auf der Client-Seite. Im X-Window System ist es die Applikation auf dem Client, die auf Ressourcen zugreift, welche ihr vom Server, also dem PC, der über Eingabe und Ausgabe Devices verfügt, geliefert wird. Hier wird die Seite, an der sich in der Regel der Benutzer befindet, als Server bezeichnet.

Abbildung 4-2. Das X-Window-Client-Server-System



[Zurück](#)
Subframing / Verfolgung

[Zum Anfang](#)

[Nach vorne](#)
X-Window-Nachrichten

X-Window-Nachrichten

Es gibt vier Arten von Nachrichten die durch das X-Protokoll zwischen Client und Server ausgetauscht werden.

- Ein *request* (Anfrage) wird meistens von der Xlib an den Server versendet oder seltener von der Xlib an sich selbst. Die Vielzahl von Nachrichten, die an den Server gerichtet sind, dienen zur Übertragung von Informationen wie beispielsweise die Aufforderung einen Punkt zu zeichnen, das Festlegen eines Fenster-Hintergrund-Bildes oder die Abfragen der Fensterabmessungen. Nachrichten, die an das eigene oder an andere Xlib-Programme gerichtet sind, dienen zur Veränderung und Abfrage von lokalen Informationen und treten eher selten auf.
- Ein *reply* (Antwort) tritt bei Anfragen an den Server auf, die eine Antwort benötigen. Eine solche Anfrage wird auch *Round-trip* Anfrage genannt, da nach dem Absetzen der Anfrage der Programmfluss unterbrochen wird, um auf eine Antwort zu warten. Es ist empfehlenswert, solche Anfragen zu vermeiden, da sie die durch das Warten auf Nachrichten, die Performance der Applikation verschlechtern. Alternativ gibt es auch Anfragen, die ein *reply* anfordern, welches zu einem späteren Zeitpunkt von einer anderen Xlib-Methode aus dem Eingangspuffer gelesen wird.
- Ein *event* (Ereignis) tritt auf, wenn ein an den Server angeschlossenes Eingabegerät neue Informationen bereitstellt (z.B. durch einen Tastendruck auf der Tastatur) oder, wenn als Nebeneffekt zu einer Vorhergehenden Anfrage ein Event erzeugt wurde. Die *events* werden auf der Xlib Seite in einem Eingabepuffer zwischengespeichert und können einzeln daraus gelesen werden. Der Client kann dem Server mitteilen, welche *events* der Server an ihn übertragen soll, womit das Nachrichtenaufkommen zwischen Client und Server gering gehalten werden kann.
- Die Aufgabe des *error* (Fehler) ist es, dem Client mitzuteilen, dass ein vorhergehender Aufruf fehlerhaft war. In der Xlib der Anwendungen werden *errors* von einer speziellen Fehlerbehandlungsroutine empfangen. Standardmäßig erfolgt eine Ausgabe des Fehlers und das Programm wird beendet. Es ist möglich diese Routine durch eigenen Quellcode zu ersetzen.

Nachrichten werden nicht direkt von den Clients zu dem Server versendet, sondern erst zwischengespeichert. Ein Vorteil dieses Verfahrens ist, dass nicht jede Nachricht einzeln durch die Netzwerkprotokolle verpackt wird, sondern dass sich immer mehrere solcher X-Protokoll-Nachrichten in einem Netzwerk-Protokoll-Paket befinden. Dadurch wird weniger Protokollinformation über das Netzwerk versendet. Ein anderer wichtiger Vorteil dieser Technik ist, dass die Applikation nicht warten muss bis eine Nachricht über das Netzwerk gesendet wird, sondern sofort ungehindert weiter arbeiten kann. Es existieren Zwischenspeicher sowohl für die eingehenden als auch für die ausgehenden Nachrichten auf der Client-Seite. Auf der Server-Seite werden die Nachrichten vom Server zum Client in der Regel sofort gesendet. Lediglich die eingehenden Nachrichten werden in einem Zwischenspeicher bis zu ihrer Abarbeitung gespeichert.

Um die gesammelten Nachrichten auf der Client-Seite zu versenden, muss eine der folgenden Bedingungen eintreten:

- Eine Applikation ruft eine Xlib-Funktion auf, die eine Nachricht erwartet, die nicht oder noch nicht eingetroffen ist. Da die Applikation nicht weiter arbeiten kann, wird diese Leerlaufzeit zum Senden

X-Window-Nachrichten

der Nachrichten verwendet.

- Das Versenden wird gezielt ausgelöst. Gerade bei der Beispielapplikation, wo sich das Fenster ohne zutun des Benutzers ständig verändert, würde das Warten auf die erste Bedingung zu lange dauern.

Auf der Client-Seite können die eingehenden Nachrichten durch Methoden der Xlib in ihrer Eingangsreihenfolge aus dem Zwischenspeicher gelesen werden. Da sich mehrere Ereignisse in dem Zwischenspeicher befinden können, muss dieser bis zur vollständigen Abarbeitung in einer Schleife zyklisch abgefragt werden. In vielen Anwendungen geschieht dies durch einen Xlib-Aufruf, in einer Endlosschleife, der den Programmablauf anhält, wenn kein weiteres Ereignis im Zwischenspeicher zur Abarbeitung bereit steht. Diese Endlosschleife verharrt so lange in der blockierenden Xlib-Anweisung bis ein neues Ereignis eintritt. Wurde dieses Ereignis abgearbeitet, kehrt die Schleife wieder in den Wartezustand zurück. Da es in der Beispielapplikation nicht denkbar wäre den Programmablauf zu unterbrechen, wird hier nach jedem abgearbeiteten Frame überprüft, ob sich eine Nachricht in Nachrichtenspeicher befindet und diese gegebenenfalls bearbeitet. Liegt keine Nachricht an, wird der Programmfluss sofort fortgesetzt.

Um das Nachrichtenaufkommen gering zu halten, muss bei der Initialisierung spezifiziert werden, welche Nachrichten überhaupt erst in den Zwischenspeicher aufgenommen werden.

Zurück

Grafisches-Fenster

Zum Anfang

Nach oben

Nach vorne

Xlib-Fenster

Xlib-Fenster

Verwendet die Client-Applikation ein grafisches Fenster, werden folgende Anweisungen zur Erzeugung des Fensters abgearbeitet:

- Mit der Methode `XOpenDisplay()` Verbindung zum X-Server aufbauen.
- Mit den eingeholten Informationen über den physikalischen Bildbereich wird die Fenstergröße für das Ausgabefenster errechnet. In der Beispielapplikation ist diese Größe durch die Auflösung des Capturebereiches vorbestimmt.
- Das Fenster wird mit `XCreateSimpleWindow()` angelegt.
- Sollen Tastatur oder Mauseingaben verarbeitet werden, ist es nötig, diese Ereignisse zu spezifizieren.
- Sollen Textausgaben in das Fenster erfolgen, muss ein entsprechender Zeichensatz geladen werden.
- Erzeugung eines Grafik-Kontext, um in dem Frame arbeiten zu können.
- Den Inhalt des Fensters bestimmen oder erzeugen.
- Das Fenster mit `XMapWindow()` anzeigen.
- Werden Tastatur oder Mauseingaben verarbeitet, ist das Abfangen und Bearbeiten dieser Ereignisse notwendig.

Die Methoden der Klasse `video_out` in der Beispielapplikation implementieren diese Verfahren. Der Konstruktor `video_out::video_out` ist hierbei für den Verbindungsaufbau, das Einholen der Fensterinformationen, das Anlegen des Fensters, die Spezifizierung der zu überwachenden Ereignisse und die Erzeugung des Grafik Kontext zuständig. Die beiden Methoden `video_out::display_frame` sind dafür verantwortlich, dass sich im Speicher befindliche Bild an ein `XImage` Format zu binden und in den Frame zu schreiben. Diese Methode ist zweifach implementiert, damit sowohl das eingefangene RGB-Bild als auch das beim Threshold und der Segmentierung erzeugte aus Integerwerten bestehende Graustufenbild, dargestellt werden kann. Beendet wird diese Methode durch den Befehl `XFlush`. Er sorgt dafür, dass die Veränderungen im Frame sofort dem Server mitgeteilt werden und nicht zu lange im Nachrichtenspeicher verweilen. Von der Methode `video_out::get_event` wird die Ereignisbehandlung durchgeführt.

Zum Vergleich und für weitere Informationen zum X Window System sei hier auf die Manpage (Anleitung in einem speziellen elektronischen Format) zu X und auf die Quelle [Nye1992] verwiesen.

Da die Darstellung eines grafischen Fensters ein rechenzeitaufwendiges Unterfangen ist und durch sie Abhängigkeiten zwischen der Applikation und dem GUI entstehen, ist es empfehlenswert, diese Ausgabemöglichkeit nur für Anschauungszwecke zu verwenden. In der Praxis würde man die durch den Erkennungs- / Verfolgungsprozess erlangten Informationen beispielsweise direkt in einer Datenbank ablegen oder über eine Netzwerkkarte an einen anderen Rechner übertragen.

Kapitel 5. Diskussion und Bewertung

Inhaltsverzeichnis

[System-Auslastung](#)[Methoden-Auslastung](#)[Algorithmen](#)[Tests](#)[Weiterentwicklung und Verbesserung](#)

System-Auslastung

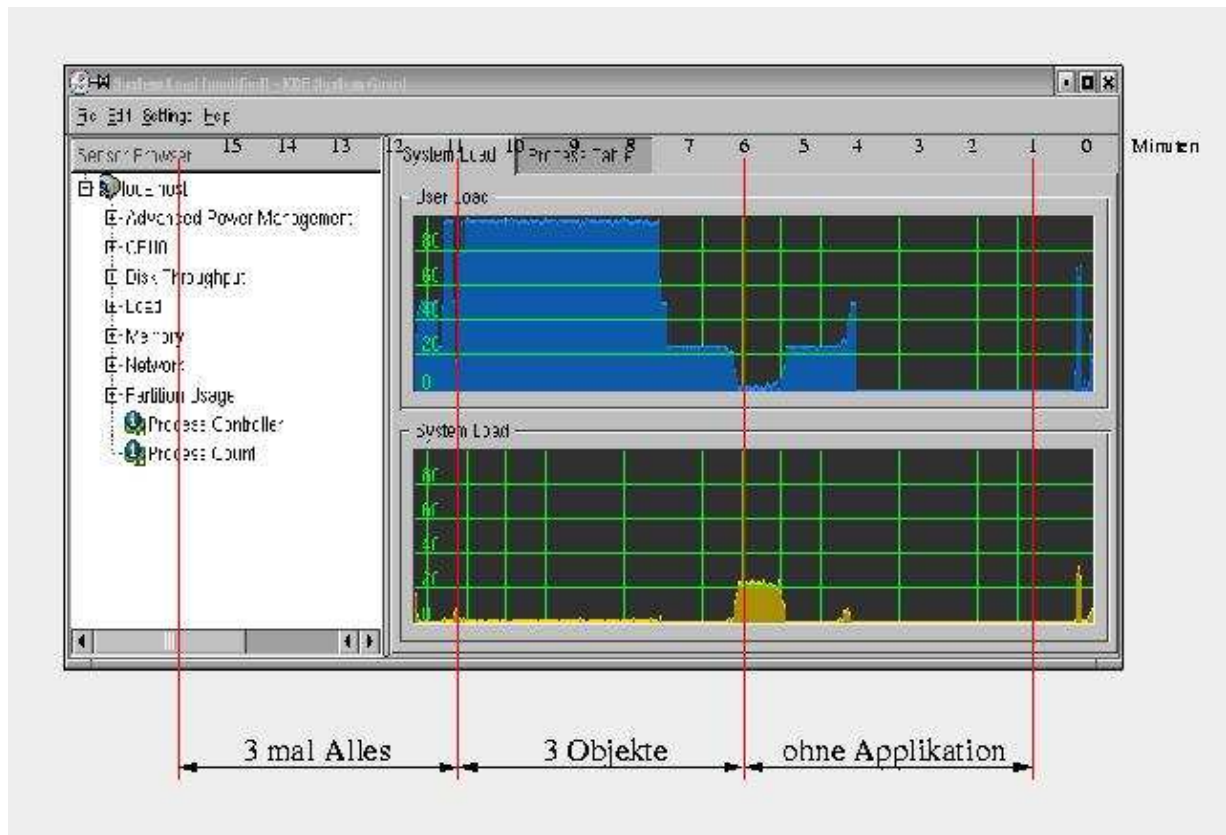
Zu Beginn der Diplomarbeit war noch nicht sichergestellt, dass sich eine kameragestützte Objektverfolgung unter Linux, wie sie z.B. für die Beobachtung der Rennwagen auf der Rennstrecke benötigt wird, überhaupt realisieren lässt. Betrachtet man jetzt die Systemauslastung durch die Beispielapplikation, erkennt man, dass eine Objektverfolgung mit 25 Bildern pro Sekunde auf einen PC System mit 800 MHz durchaus möglich ist. Dabei rührt die Begrenzung auf 25 Bilder pro Sekunde von den Eigenschaften der verwendeten *Frame-Grabber-Karte* in Verbindung mit der Kamera. Im Normalfall wird ca. 20-50% der Systemleistung für die Erkennung/Verfolgung eines Objektes benötigt. Bei besonders ungünstigen Objekten und Thresholdfiltern kann es jedoch vorkommen, dass die Systemrechenleistung nicht ausreicht und Frames verloren gehen.

Betrachtet man die *System und Benutzer Last* für die Fälle,

- dass die Beispielapplikation dreimal das komplette Bild beobachtet (siehe Abb. *drei mal das gesamte Bild*),
- dass die Beispielapplikation drei Objekte in "Subframes" verfolgt (siehe Abb. *drei Papierkugeln in Subframes*) und
- dass die Beispielapplikation beendet wurde,

so ergibt sich folgende Auslastung:

Abbildung 5-1. System und Benutzer Last



Alle Tests wurde mit einer Auflösung von 320 x 240 Bildpunkten durchgeführt.

Bei dieser Abbildung (Abb. *System und Benutzer Last*) wird die Auslastung in "System-Load" und "User-Load" unter Verwendung einer an die *Frame-Grabber-Karte* angeschlossenen Kamera angegeben. Load ist hierbei ein Maß dafür, wie viele Prozesse bereit sind die CPUs zu nutzen, geteilt durch die Anzahl der im System zur Verfügung stehenden CPUs. Bei einem Load unter 100 Prozent sind die CPUs in der Lage innerhalb eines Zeitabschnittes alle anstehenden Prozesse abzuarbeiten. Je weiter sich dieser Wert der 100 Prozent Grenze nähert, desto stärker ist die CPU ausgelastet. Die Unterscheidung zwischen "System-Load" und "User-Load" beruht darauf von wem die CPU beansprucht wird. Die Beispielapplikation beispielsweise erzeugt hauptsächlich "User-Load" und nur sehr wenig "System-Load". "System-Load" wird von der Beispielapplikation z.B. durch das Aufrufen von *System-Calls* erzeugt. Natürlich erzeugt das Betriebssystem selbstständig eine oft ungleichmässige Menge an "System Load".

Bei der Abbildung der Lasten mit KDE System Guard 1.1.0 fällt auf, dass teilweise User-Load zu System-Load wird. Dieses Verhalten ist wahrscheinlich auf eine Fehlzuordnung zurückzuführen, deshalb ist es sinnvoll in diesen Fällen das "System-Load" dem "User-Load" anzurechnen.

In der Abbildung ist zu erkennen, dass, wenn die *drei Papierkugeln in Subframes* verfolgt werden, die CPU zu ca. 20-50% ausgelastet ist. Bei einer solchen Belastung lässt sich die Höchststrate von 25 Bildern pro Sekunde erreichen. Wird hingegen der Thresholdwert so ungeschickt gewählt, dass *drei mal das gesamte Bild* bearbeitet werden muss, ist die CPU maximal ausgelastet bzw. überlastet. In diesem Fall sinkt die Frame Rate auf 5-6 Bilder pro Sekunde.

Zwischen maximaler Auslastung und Überlastung lässt sich anhand dieser Grafik nicht unterscheiden, da keine Werte oberhalb der 100 % Grenze angezeigt werden. Wird das System im Ruhezustand ohne Beispielapplikation betrachtet, erkennt man wie andere Prozesse, die unter dem selben User laufen, ein

Minimum an Systemleistung verbrauchen.

Abbildung 5-2. drei Papierkugeln in Subframes

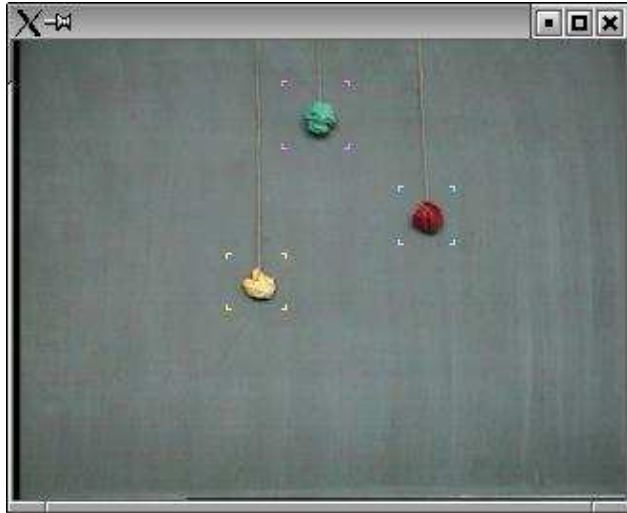
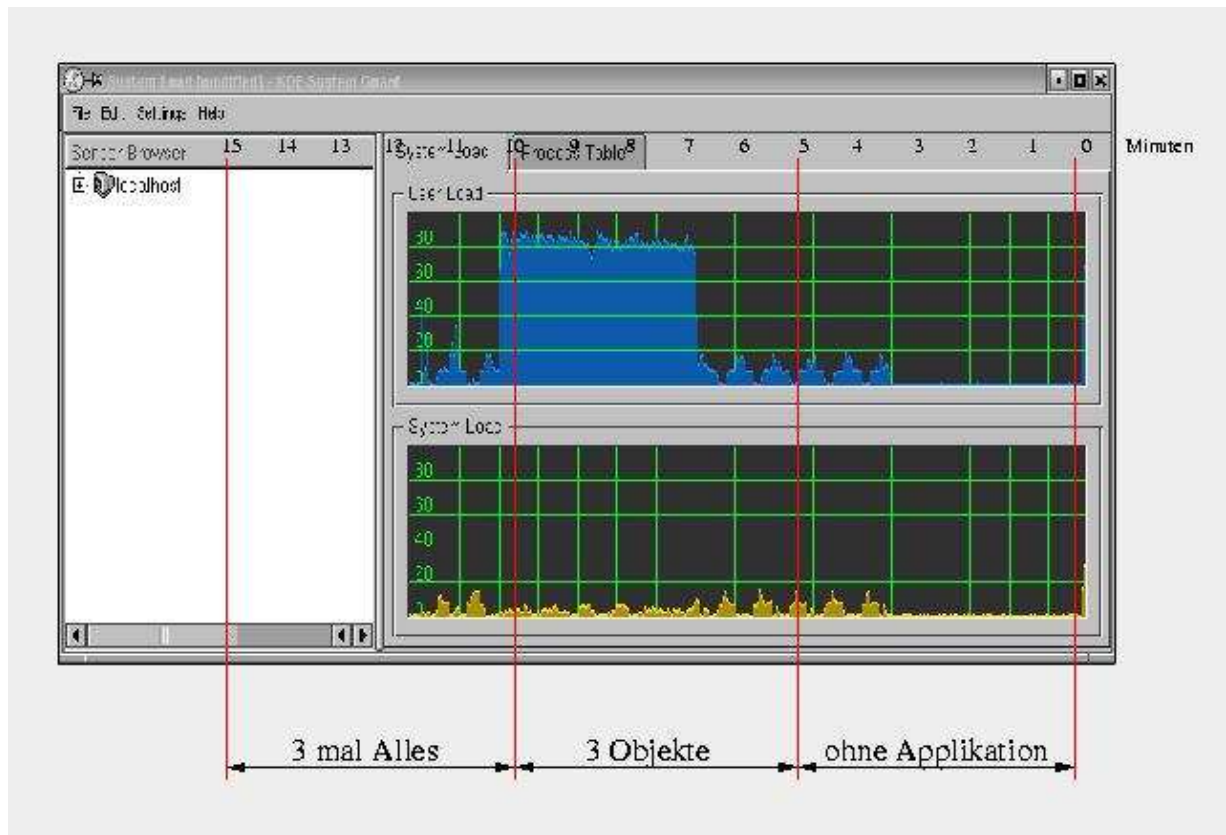


Abbildung 5-3. drei mal das gesamte Bild



Betrachtet man die Auslastung für die Bildaufnahme mit der "TerraTech USB Pro" Webcam, die mit einem C-MOS Sensor ausgestattet ist, ergibt sich für den Fall, dass drei Objekte im Subframes verfolgt werden eine Auslastung von ca. 18 Prozent und eine Framerate von 8 Bildern pro Sekunde. Muss aufgrund schlechter Wahl der Thresholdwerte das komplette Bild dreifach bearbeitet werden ist das System zu 80 bis 85 Prozent ausgelastet und erreicht nur noch eine Framerate von ca. 5 Bildern pro Sekunde (siehe Abb. System und Benutzer Last bei Verwendung der USB-Kamera).

Abbildung 5-4. System und Benutzer Last bei Verwendung der USB-Kamera



Es fällt auf, dass die Framerate bei Verwendung der "TerraCam USB Pro" Webcam geringer sind als bei Einsatz einer Kamera Frame-Grabber-Karte Kombination. Betrachtet man jedoch einen im April 2002 erschienenen Artikel des Magazins c't über USB-Kameras [Peeck2002] so lässt sich erkennen, dass andere USB-Kamera-Modelle in der Lage sind Frame Raten von bis zu 30 Bildern pro Sekunde umzusetzen, während die Ergebnisse für die "TerraCam USB Pro" mit 15 Bildern pro Sekunde, bei einer Auflösung von 320 x 240 Bildpunkten und 7 Bilder pro Sekunde, bei 640 x 480 Bildpunkten angegeben sind. Leider beziehen sich die dortigen Angaben auf das Betriebssystem "Microsoft Windows ME" und "Microsoft Windows XP" der Firma "Microsoft". Betrachtet man jedoch z.B. die "ToUCam Pro" von Philips mit 30 Bildern pro Sekunde, bei einer Auflösung von 320 x 240 Bildpunkten und 29 Bilder pro Sekunde, bei 640 x 480 Bildpunkten sowie die Tatsache, dass Philips Treiber für Linux anbietet, besteht die Hoffnung, dass sich durch die Wahl einer anderen USB-Kamera bessere Frame Raten erzielen lassen.

Aufgrund der Optik der USB-Kamera ist das Testszenario geringfügig anders, als bei der an die Frame-Grabber-Karte angeschlossenen Kamera (siehe Abb. drei Papierkugeln in Subframes mit einer USB-Kamera aufgenommen und Abb. drei Mal das gesamte Bild mit einer USB-Kamera aufgenommen).

Abbildung 5-5. drei Papierkugeln in Subframes mit einer USB-Kamera aufgenommen



Abbildung 5-6. drei Mal das gesamte Bild mit einer USB-Kamera aufgenommen



Dadurch, dass die Load-Darstellung nicht nur von der Beispielapplikation abhängt, sondern auch durch, mit dieser Applikation konkurrierende Prozesse, beeinflusst werden, können die dort dargestellten Werte nicht die Beispielapplikation hundertprozentig repräsentieren. Für eine grobe Betrachtung der Rechenkernauslastung durch die Beispielapplikatin ist diese Darstellung jedoch ausreichend.

[Zurück](#)
Xlib-Fenster

[Zum Anfang](#)

[Nach vorne](#)
Methoden-Auslastung

Methoden-Auslastung

Möchte man genauer untersuchen wie sich Rechenleistung innerhalb einer Applikation verteilt, oder wie oft eine Methode aufgerufen wird, so ist es möglich die Applikation mit der Compiler-Option "-gp" und je nach gcc Version dem Compilerparameter "-fprofile-arcs" zu compilieren (siehe Manpage zu gprof). Wird die Applikation nun erfolgreich ausgeführt und beendet, kann mit Hilfe der Programme gprof oder kprof eine von der Applikation erzeugte Profile-Datei ausgewertet werden.

Im Folgenden sind Auszüge von Profilen dargestellt. Bei den Auszügen wurden die Parameter der Methoden entfernt, um eine höhere Übersichtlichkeit zu erhalten. Alle Profile wurden während einer 120 sekundigen Laufzeit der Beispielapplikation erzeugt.

Die Überschriften der einzelnen Profil-Spalten bedeuten:

- % time

Der prozentuale Anteil der Programm-Laufzeit, welche diese Methode verbraucht hat.

- cumulative seconds

Die fortlaufende Anzahl der Sekunden dieser Methode und der ihr übergeordneten Methoden.

- self seconds

Die Anzahl der Sekunden, die für diese Methode als Laufzeit erfasst wurden. Die Tabelle ist nach diesem Wert sortiert.

- calls

Die Anzahl der Aufrufe dieser Funktion.

- self ms/call

Der Mittelwert der Sekunden, die in diese Methode pro Aufruf verbracht wurden.

- total ms/call

Der Mittelwert der Sekunden, die in diese Methode und ihren untergeordneten Methoden pro Aufruf verbracht wurden.

- name

Der Name der Funktion.

Beim ersten Durchlauf (siehe *Profil-Informationen (Auszug) der Beispielapplikation*) wurden die drei verschiedenfarbigen Papierkugeln wie sie schon im Abschnitt *System-Auslastung* verwendet wurden, je einzeln in einen Subframe verfolgt (siehe Abb. *drei Papierkugeln in Subframes*).

Betrachtet man die Gesamtlaufzeit der Applikation von 120 Sekunden im Vergleich zur Gesamtlaufzeit der einzelnen Methoden von 30.53 Sekunden, stellt man fest, dass die Applikation nur $30.53s / 120s * 100\% = 25.44\%$ aktiv ist. Das meiste der restlichen Zeit verbringt sie mit dem Warten auf den nächsten Frame, soweit

Methoden-Auslastung

ihr keine andere Applikation die Rechenzeit streitig macht.

Wie schon zuvor in dieser Arbeit erwähnt, ist es sehr gewagt, eine Applikation an ein GUI zu binden. Man erkennt, dass die Methode `video_out::display_frame` 95.45 Prozent der Gesamtlaufzeit benötigt, um die Bilder in das GUI zu schreiben. Es ist also sehr empfehlenswert, auf eine grafische Ausgabe zu verzichten und zum Beispiel die Position der verfolgten Objekte in eine Datenbank abzulegen, oder über ein Netzwerk an andere Rechner zu übertragen. Die restlichen 4,55 % übernimmt hauptsächlich die Methode `video_out::threshold` mit 4.42 %, da sie aus dem im RGB-Format-Bild lesen muss und mit den erhaltenen Informationen das Thresholding durchführt. Aufgrund der Tatsache, dass durch das Subframing nur sehr wenig an Pixeln übrig bleibt, mit denen gearbeitet werden muss, treten Methoden, die von der Anzahl der Vordergrund-pixelzahl unabhängig sind, an den dritten und vierten Platz, mit Rechenzeitanteilen von 0.07 % für die Methode `video_in::grab_pix`, 0.03 % für `video_in::grab_frame` und 0.03 % für die Methode `main`. Der Rest weist einen so geringe Laufzeit auf, dass gprof keine Zeit erfassen konnte.

Entsprechend zu den kleinen Betrachtungsbereichen, die durch das Subframing erzeugt wurden, sind die Aufrufzahlen der Vordergrund-Pixel abhängigen Methoden `video_in::just_count` und `video_in::labeling` mit je 772.258 Aufrufen selten. Diese Zahl ist gering, da bei einer Auflösung von $320 * 240$ Pixeln und 3005 gegrabten Bildern (wie z.B. an der Methode `video_in::grab_frame` zu sehen), für drei zu verfolgende Objekte, im schlimmsten Fall $320 * 240 * 3005 * 3 = 692.352.000$ Punkte verarbeitet werden.

Eine interessante Betrachtung zum Überprüfen der Funktion der Applikation ergibt sich auch aus den Aufrufverhältnissen. Da drei Instanzen der Klasse `o_tracing` für die Verfolgung von drei Objekten verwendet werden, wird der Konstruktor `o_tracing::o_tracing` drei mal aufgerufen. Ein anderes Beispiel ist, dass eins zu drei Verhältnis zwischen der Methode `video_in::grab_frame`, die für jedes einzufangende Bild aufgerufen wird und der Methode `o_tracing::draw_tracing_frame`, welche für jeden der drei Sufame die Eck-Markierungen in das grafische Fenster malt.

Abbildung 5-7. Profil-Informationen (Auszug) der Beispielapplikation

Flat profile:

Each sample counts as 0.01 seconds.

%	cumulative	self		self	total	
time	seconds	seconds	calls	ms/call	ms/call	name
95.45	29.14	29.14	3005	9.70	9.70	<code>video_out::display_frame()</code>
4.42	30.49	1.35	9015	0.15	0.15	<code>o_tracing::threshold()</code>
0.07	30.51	0.02	3005	0.01	0.01	<code>video_in::grab_pix()</code>
0.03	30.52	0.01	3005	0.00	0.00	<code>video_in::grab_frame()</code>
0.03	30.53	0.01				<code>main</code>
0.00	30.53	0.00	772258	0.00	0.00	<code>o_tracing::just_count()</code>
0.00	30.53	0.00	772258	0.00	0.00	<code>o_tracing::labeling()</code>
0.00	30.53	0.00	77961	0.00	0.00	<code>o_tracing::in_label()</code>
0.00	30.53	0.00	60155	0.00	0.00	<code>o_tracing::create_roots()</code>
0.00	30.53	0.00	9015	0.00	0.00	<code>o_tracing::draw_tracing_frame()</code>
0.00	30.53	0.00	9015	0.00	0.00	<code>o_tracing::join_labels()</code>
0.00	30.53	0.00	9015	0.00	0.15	<code>o_tracing::start_tracing()</code>
0.00	30.53	0.00	9015	0.00	0.00	<code>o_tracing::tracing(void)</code>
0.00	30.53	0.00	3005	0.00	0.00	<code>Menue()</code>
0.00	30.53	0.00	3005	0.00	0.00	<code>video_out::get_event()</code>
0.00	30.53	0.00	3005	0.00	0.00	<code>video_in::grab_sync()</code>
0.00	30.53	0.00	3	0.00	0.00	<code>o_tracing::o_tracing()</code>
0.00	30.53	0.00	3	0.00	0.00	<code>video_out::handle_button_down()</code>

Methoden-Auslastung

0.00	30.53	0.00	3	0.00	0.00	<code>o_tracing::set_rgb_threshold()</code>
0.00	30.53	0.00	3	0.00	0.00	<code>o_tracing::set_total_threshold()</code>
0.00	30.53	0.00	3	0.00	0.00	<code>settoleranz()</code>
0.00	30.53	0.00	1	0.00	0.00	<code>o_tracing::~~o_tracing()</code>
0.00	30.53	0.00	1	0.00	0.00	<code>video_in::video_in()</code>
0.00	30.53	0.00	1	0.00	0.00	<code>video_out::video_out()</code>
0.00	30.53	0.00	1	0.00	0.00	<code>video_out::create_simple_window()</code>
0.00	30.53	0.00	1	0.00	0.00	<code>video_out::get_depth()</code>
0.00	30.53	0.00	1	0.00	0.00	<code>video_in::grab_close()</code>
0.00	30.53	0.00	1	0.00	0.00	<code>video_in::grab_frame()</code>
0.00	30.53	0.00	1	0.00	0.00	<code>video_in::grab_open()</code>

Eine noch interessantere Betrachtung ergibt sich, wenn die Applikation größere Subframes bearbeiten muss (siehe Abb. *Profil-Information (Auszug) der Beispielapplikation bei schlechter Wahl der Threshold-Level*). Bei diesem Durchlauf wurde der Thresholdwert so schlecht gewählt, dass drei Mal das komplette Bild bearbeitet wird. Eine entsprechende Aufnahme des Bildes befindet sich im Abschnitt *System-Auslastung* (siehe Abb. *drei Papierkugeln in Subframes*).

Betrachtet man wieder die Gesamtlaufzeit der Applikation von 120 Sekunden im Vergleich zur Gesamtlaufzeit der einzelnen Methoden von diesmal 86.04 Sekunden, stellt man fest, dass die Applikation mit $86.04s / 120s * 100\% = 71.70\%$ erheblich mehr aktiv ist, als bei der vorhergehenden Messung.

Die meiste Zeit verbringt die Applikation jetzt mit dem Erkennen von Flächen in der Methode `o_tracing::labeling` mit 39,46 % der Gesamtlaufzeit und mit 23.19 % in der Methode `o_tracing::just_count`, die für das Eintragen der Vordergrund-Pixel in die Labels-Tabelle zuständig ist. Diese hohen Laufzeiten ergeben sich hauptsächlich daher, dass beide Methoden 145.793.470 mal aufgerufen werden. Diese Zahl ist sehr hoch im Verhältnis zum schlimmsten Fall, dass jedes Pixel bearbeitet werden muss. Bei einer Auflösung von $320 * 240$ Bildpunkten, in 699 aufgenommenen Bildern, mit je drei Subframes, ergeben sich im schlimmsten Fall $320 * 240 * 699 * 3 = 161.049.600$ zu bearbeitende Bildpunkte. Den dritten Platz belegt nun die Methode `o_tracing::threshold` mit einem Anteil von 22,33 % an der Gesamtlaufzeit, da von ihr ohne brauchbares Subframing eine erheblich grössere Fläche bearbeitet werden muss. Die Menge der zu verarbeitenden Vordergrund-Pixel und die grossen Flächen bewirken auch, dass die Methode `o_tracing::join_labels` 10,26 %, die Methode `o_tracing::in_label` 0.86 % und die Methode `o_tracing::create_roots` 0.12 % der Gesamtlaufzeit verbrauchen. Dass die Methode `video_out::display_frame`, die für die Darstellung des Ausgabebildes im *GUI* zuständig ist, mit 3,74 % verhältnismäßig weniger Rechenleistung benötigt, wäre verständlich. Seltsamerweise wird diese Ausgabe jedoch unter Belastung schneller. So verbrauchte die Methode im ersten Durchlauf 9.7 ms für einen Aufruf und benötigte im zweiten Durchlauf, bei Belastung der Applikation, nur 4.61 ms pro Aufruf.

Die restlichen Methoden verbrauchen nur so wenig Laufzeit, dass sie nicht weiter betrachtet werden. Bei der Betrachtung der Verhältnisse ändert sich zwischen dem ersten und zweiten Durchlauf erwartungsgemäß nichts. Seltsam ist, dass die Methode `main` aus der Liste der Methoden, bei der Aufnahme mit sehr grossen Subframes, einfach verschwindet.

Abbildung 5-8. Profil-Information (Auszug) der Beispielapplikation bei schlechter Wahl der Threshold-Level

Flat profile:

Methoden-Auslastung

Each sample counts as 0.01 seconds.

%	cumulative	self	self	self	total	
time	seconds	seconds	calls	ms/call	ms/call	name
39.46	33.95	33.95	145793470	0.00	0.00	o_tracing::labeling()
23.19	53.90	19.95	145793470	0.00	0.00	o_tracing::just_count()
22.33	73.11	19.21	2097	9.16	35.22	o_tracing::threshold()
10.26	81.94	8.83	2097	4.21	4.26	o_tracing::join_labels()
3.74	85.16	3.22	699	4.61	4.61	video_out::display_frame()
0.86	85.90	0.74	1171811	0.00	0.00	o_tracing::in_label()
0.12	86.00	0.10	190736	0.00	0.00	o_tracing::create_roots()
0.01	86.01	0.01	2097	0.00	0.00	o_tracing::draw_tracing_frame()
0.01	86.02	0.01	2097	0.00	39.48	o_tracing::start_tracing()
0.01	86.03	0.01	2097	0.00	0.00	o_tracing::tracing()
0.01	86.04	0.01	699	0.01	0.01	Menue()
0.00	86.04	0.00	699	0.00	0.00	video_out::get_event()
0.00	86.04	0.00	699	0.00	0.00	video_in::grab_frame()
0.00	86.04	0.00	699	0.00	0.00	video_in::grab_pix()
0.00	86.04	0.00	699	0.00	0.00	video_in::grab_sync()
0.00	86.04	0.00	3	0.00	0.00	o_tracing::o_tracing()
0.00	86.04	0.00	3	0.00	0.00	video_out::handle_button_down()
0.00	86.04	0.00	3	0.00	0.00	o_tracing::set_rgb_threshold()
0.00	86.04	0.00	3	0.00	0.00	o_tracing::set_total_threshold()
0.00	86.04	0.00	3	0.00	0.00	settoleranz()
0.00	86.04	0.00	1	0.00	0.00	o_tracing::~~o_tracing()
0.00	86.04	0.00	1	0.00	0.00	video_in::video_in()
0.00	86.04	0.00	1	0.00	0.00	video_out::video_out()
0.00	86.04	0.00	1	0.00	0.00	video_out::create_simple_window()
0.00	86.04	0.00	1	0.00	0.00	video_out::get_depth()
0.00	86.04	0.00	1	0.00	0.00	video_in::grab_close()
0.00	86.04	0.00	1	0.00	0.00	video_in::grab_frame()
0.00	86.04	0.00	1	0.00	0.00	video_in::grab_open()

Die selben Betrachtungen mit einer USB-Kamera durchgeführt, ergeben unter Berücksichtigung der geringeren Frameraten, ähnliche Ergebnisse (siehe Abb. Profil-Informationen (Auszug) der Beispielapplikation Beispielapplikation bei einer USB-Kamera und Abb. Profil-Information (Auszug) der Beispielapplikation bei schlechter Wahl der Threshold-LevelThreshold-wert und USB-Kamera). Für die erste Messung wurden die USB-Testobjekte wie schon in Abschnitt System-Auslastung mit Subframing verwendet (siehe Abb. drei Papierkugeln in Subframes mit einer USB-Kamera aufgenommen). Für die zweite Messung fand das gleiche Szenario mit einer bewusst schlechten Wahl der Thresholdwerte seine Anwendung (siehe Abb. drei Mal das gesamte Bild mit einer USB-Kamera aufgenommen).

Abbildung 5-9. Profil-Informationen (Auszug) der Beispielapplikation bei einer USB-Kamera

Flat profile:

Each sample counts as 0.01 seconds.

%	cumulative	self	self	self	total	
time	seconds	seconds	calls	ms/call	ms/call	name
89.47	4.42	4.42	989	4.47	4.47	video_out::display_frame()
6.48	4.74	0.32	2967	0.11	0.14	o_tracing::threshold()
1.62	4.82	0.08	2967	0.03	0.03	o_tracing::join_labels()
1.21	4.88	0.06	335128	0.00	0.00	o_tracing::labeling()
0.61	4.91	0.03	335128	0.00	0.00	o_tracing::just_count()
0.40	4.93	0.02	17396	0.00	0.00	o_tracing::create_roots()

Methoden-Auslastung

0.20	4.94	0.01	2967	0.00	0.00	o_tracing::draw_tracing_frame()
0.00	4.94	0.00	46106	0.00	0.00	o_tracing::in_label()
0.00	4.94	0.00	2967	0.00	0.17	o_tracing::start_tracing()
0.00	4.94	0.00	2967	0.00	0.00	o_tracing::tracing()
0.00	4.94	0.00	989	0.00	0.00	Menue()
0.00	4.94	0.00	989	0.00	0.00	video_out::get_event()
0.00	4.94	0.00	989	0.00	0.00	video_in::grab_frame()
0.00	4.94	0.00	989	0.00	0.00	video_in::grab_pix()
0.00	4.94	0.00	989	0.00	0.00	video_in::grab_sync()
0.00	4.94	0.00	4	0.00	0.00	video_out::handle_button_down()
0.00	4.94	0.00	4	0.00	0.00	o_tracing::set_rgb_threshold()
0.00	4.94	0.00	4	0.00	0.00	settoleranz()
0.00	4.94	0.00	3	0.00	0.00	o_tracing::o_tracing()
0.00	4.94	0.00	3	0.00	0.00	o_tracing::set_total_threshold()
0.00	4.94	0.00	1	0.00	0.00	o_tracing::~~o_tracing()
0.00	4.94	0.00	1	0.00	0.00	video_in::video_in()
0.00	4.94	0.00	1	0.00	0.00	video_out::video_out()
0.00	4.94	0.00	1	0.00	0.00	video_out::create_simple_window()
0.00	4.94	0.00	1	0.00	0.00	video_out::get_depth()
0.00	4.94	0.00	1	0.00	0.00	video_in::grab_close()
0.00	4.94	0.00	1	0.00	0.00	video_in::grab_frame()
0.00	4.94	0.00	1	0.00	0.00	video_in::grab_open()

Abbildung 5-10. Profil-Information (Auszug) der Beispielapplikation bei schlechter Wahl der Threshold-Level und USB-Kamera

Flat profile:

Each sample counts as 0.01 seconds.

%	cumulative	self		self	total	
time	seconds	seconds	calls	ms/call	ms/call	name
37.79	28.66	28.66	121906710	0.00	0.00	o_tracing::labeling()
22.72	45.89	17.23	121906710	0.00	0.00	o_tracing::just_count()
21.48	62.18	16.29	1797	9.07	36.18	o_tracing::threshold()
10.01	69.77	7.59	1797	4.22	4.33	o_tracing::join_labels()
3.98	72.79	3.02	599	5.04	5.04	video_out::display_frame()
3.74	75.63	2.84	1379239	0.00	0.00	o_tracing::in_label()
0.25	75.82	0.19	325060	0.00	0.00	o_tracing::create_roots()
0.01	75.83	0.01	1797	0.01	0.01	o_tracing::tracing()
0.01	75.84	0.01				main
0.00	75.84	0.00	1797	0.00	0.00	o_tracing::draw_tracing_frame()
0.00	75.84	0.00	1797	0.00	40.52	o_tracing::start_tracing()
0.00	75.84	0.00	599	0.00	0.00	Menue()
0.00	75.84	0.00	599	0.00	0.00	video_out::get_event()
0.00	75.84	0.00	599	0.00	0.00	video_in::grab_frame()
0.00	75.84	0.00	599	0.00	0.00	video_in::grab_pix()
0.00	75.84	0.00	599	0.00	0.00	video_in::grab_sync()
0.00	75.84	0.00	3	0.00	0.00	o_tracing::o_tracing()
0.00	75.84	0.00	3	0.00	0.00	video_out::handle_button_down()
0.00	75.84	0.00	3	0.00	0.00	o_tracing::set_rgb_threshold()
0.00	75.84	0.00	3	0.00	0.00	o_tracing::set_total_threshold()
0.00	75.84	0.00	3	0.00	0.00	settoleranz()
0.00	75.84	0.00	1	0.00	0.00	o_tracing::~~o_tracing()
0.00	75.84	0.00	1	0.00	0.00	video_in::video_in()
0.00	75.84	0.00	1	0.00	0.00	video_out::video_out()
0.00	75.84	0.00	1	0.00	0.00	video_out::create_simple_window()
0.00	75.84	0.00	1	0.00	0.00	video_out::get_depth()
0.00	75.84	0.00	1	0.00	0.00	video_in::grab_close()
0.00	75.84	0.00	1	0.00	0.00	video_in::grab_frame()

Methoden-Auslastung

0.00 75.84 0.00 1 0.00 0.00 video_in::grab_open()

Alle Messungen in diesem Kapitel wurde mindestens zweifach durchgeführt. Bei dem Vergleichen der einzelnen Messungen einer Kategorie ergaben sich keine großen Abweichungen zwischen den hier dargestellten Ergebnissen. Durch diese Maßnahme soll vermieden werden, dass die hier dargestellten Messungen durch z.B. einen parallel laufenden Prozess oder ein gerade überlastetes GUI verfälscht werden.

Zurück

Diskussion und Bewertung

Zum Anfang

Nach oben

Nach vorne

Algorithmen

Algorithmen

Eine Objektverfolgung wie sie mit der Beispielapplikation beschrieben wird, kann nur für die Erkennung und Verfolgung einfach geformter Körper, die sich von ihrem Hintergrund abheben, verwendet werden. Da die Farbe und die Kreisflächenähnlichkeit als Attribut für die Objektidentifikation dienen, ist eine Unterscheidung zwischen ähnlichen gleichfarbigen Objekten wie beispielsweise einem Achteck und Neuneck nicht möglich.

Der Vorteil dieser Technik liegt darin, dass die verwendeten Algorithmen eine hohe Verarbeitungsgeschwindigkeit aufweisen. Betrachtet man beispielsweise zwei Modell-Rennwagen auf einer Rennstrecke, sind beide Objekte sehr ähnlich geformt. Spätestens bei der Abbildung im Speicher auf 15 Pixel je Fahrzeug wird eine Unterscheidung anhand der Form sehr schwierig. Erheblich effizienter ist es, diese Pixelanhäufungen des Rennwagens anhand seiner Farbe und Kompaktheit bzw. Kreisflächenähnlichkeit zu unterscheiden.

Eine Gefahr der verwendeten Objekterkennung ist, dass es vorkommen kann, dass das verfolgte Objekt eine andere gleichfarbige Fläche tangiert und beide zu einer neuen Fläche vereinigt werden. Hierdurch ändern sich die Eigenschaften des verfolgten Objektes, was zu einer Nichterkennung oder Fehlererkennung führen kann. Bei der Modell-Rennbahn lässt sich dies dadurch vermeiden, dass für die Rennwagen zwei unterschiedliche Farben verwendet werden, die in der unmittelbaren Umgebung der Rennstrecke nicht als größere Fläche auftauchen.

Durch die Verwendung von Methoden mit möglichst geringem Rechenleistungsbedarf, ist die Applikation in der Lage, die Objekte mit hohen Bildraten abzutasten. Würde die Verarbeitung eines Bildes länger dauern als der Zeitraum, der bis zum Eintreffen des nächsten Bildes zur Verfügung steht, gehen Bilder verloren bzw. es kann nicht die volle Framerate erreicht werden. Gerade bei der Verfolgung von schnell bewegenden Objekten, ist es wichtig, das Bild oft abzutasten, da sonst der Bereich, indem sich ein Objekt unbeobachtet bewegt, zu groß wird. Ein Modell-Rennwagen, der sich beispielsweise mit 2 Meter pro Sekunde bewegt ($v=2/s$) und mit 25 Bildern pro Sekunde abgetastet ($t=1/25s$) wird, kann sich zwischen 2 Bilder nach der Formel $s=t*v$ 8 cm unbeobachtet bewegen. Je größer diese Entfernung wird, um so größer kann auch die Abweichung von der Position ausfallen, an der das Fahrzeug durch das Subframing erwartet wird. Somit steigt die Gefahr, dass das Objekt an einem unerwarteten Punkt auftaucht, der nicht überwacht wird und so die Verfolgung (kurzzeitig) unterbrochen wird.

Bei der Wahl der Labeling Technik bestand die Möglichkeit, ob die diagonal anliegenden Nachbar-Pixel mit überprüft werden, oder ob sich der Algorithmus nur auf die links, rechts, oben und unten anliegenden Nachbarn beschränken soll (siehe Kapitel Findung der Nachbarn). Bei einigen Tests mit beiden Verfahren wurde die Technik der 8-connectivity (8 Nachbarn werden betrachtet) gewählt, da die Betrachtung von nur vier Nachbarn dazu neigt, schmale diagonale Objekte in mehrere Einzelobjekte zu zerteilen. Insgesamt zeigte der 8-connectivity Algorithmus ein stärkeres Bestreben, Flächen zu bilden und trotz Rauschen beizubehalten als die 4-connectivity Variante. Da jedoch die 4-connectivity bis zu über 50 % weniger Rechenleistung benötigt und dennoch ausreichende Ergebnisse liefert, könnte zur Optimierung auf dieses Verfahren zurückgegriffen werden.

Der grösste Anteil bei der Einsparung von Rechenzeit geht von der Verwendung eines "Subframing" aus. Diese Technik hat den Vorteil, dass für die Vorgänge des Thresholding und Labeling nur ein kleiner Bereich

Algorithmen

bearbeitet werden muss. Dies wirkt sich auch auf die Anzahl, der von der Selektion zu bearbeitender Objekte aus. So ist es möglich, in jedem dieser Bereiche Rechenleistung einzusparen. Hier muss abgewogen werden, wie klein man den Subframe wählt, ohne den Verlust des Objektes zu riskieren (siehe Kapitel Subframing / Verfolgung). Die in der Beispielapplikation verwendete Kombination aus einem dynamischen Anteil, der von der Geschwindigkeit des Objektes und der Bewegungsrichtung abhängt, gekoppelt mit einer Mindestgröße, die die doppelte Ausdehnung des Objektes umfasst, hat sich als sehr robust gegenüber Verfahren ohne Dynamik und kleiner Fenstergrösse erwiesen. Geht ein Objekt verloren, wird in der Implementierung der Beispielapplikation sofort wieder das gesamte Bild nach dem Objekt durchsucht. Da durch dieses Vergrössern des Subframe viel Rechenleistung beansprucht wird, ist es ratsam, den Subframebereich nicht zu klein zu wählen, um einem Verlust vorzubeugen.

Zurück

Methoden-Auslastung

Zum Anfang

Nach oben

Nach vorne

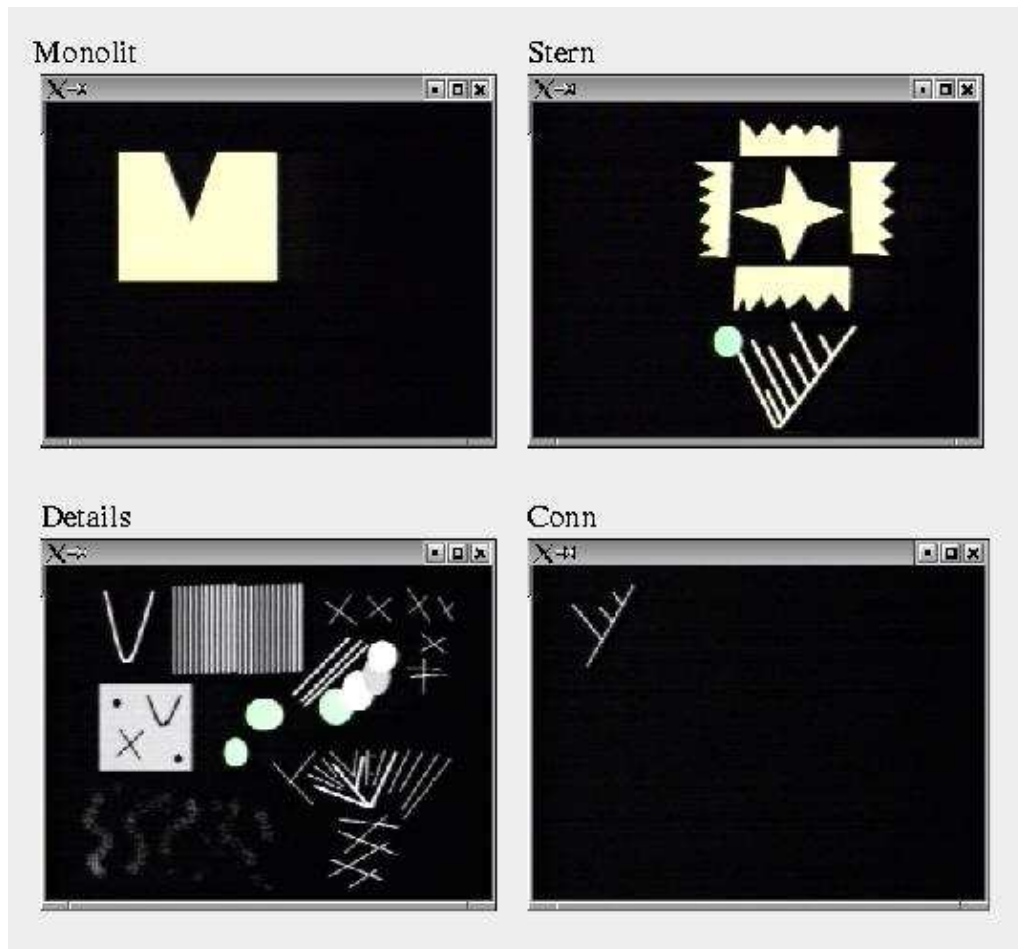
Tests

Tests

Während der Entwicklung wurde die Applikation mehrfachen Testläufen unterzogen. Zum Teil wurden Testbilder auf eine Videokassette überspielt und beim Test in die Frame-Grabber-Karte zurückgespielt. Die folgenden Bilder wurden beispielsweise zum Überprüfen des Labeling- und Threshold-Algorithmuses verwendet (siehe Threshold- und Labeling-Testframes). An dem mit "Monolit" gekennzeichneten Bild wurde in der frühen Entwicklungsphase überprüft, ob ein einfacher Körper als eine Fläche erkannt wird. Durch die große Kerbe war es möglich zu testen, ob die V-Problematik (zwei Flächen laufen zu einer zusammen) gelöst wird. Eine genauere Überprüfung der V-Problematik erfolgt in Testbild "Conn". Das gesamte Gebilde muss hierbei als eine Fläche erkannt werden.

Am Testbild "Stern" und "Details" konnte unter anderen getestet werden, ob die Objekterkennung in der Lage ist, das Objekt, das beispielsweise den größten "CONNECTIONS"-Wert hat, aus einer Menge von Objekten zu wählen.

Abbildung 5-11. Threshold- und Labeling-Testframes



Etwas aufwendiger war die Erzeugung von Testfilmen, mit deren Hilfe sich die Objektverfolgung und das Subframing überprüfen lässt (siehe *Threshold- und Labeling-Testframes*). Bei den Testfilmen "Car" und "drei Punkte" handelt es sich um mit Macromedia Flash 5 erzeugte Animationen, bei denen sich ein beziehungsweise drei Objekte auf verschlungenen Bahnen bewegen. Mit Hilfe dieser Sequenzen war es möglich zu überprüfen, ob sich Objekte sicher verfolgen lassen. Der Testaufbau "drei Kugeln" wurde speziell für die Leistungsmessung der Applikation verwendet (siehe weiter oben in diesem Kapitel).

Um sicherzustellen, dass die Applikation über längere Zeit stabil läuft, wurde die *Frame-Grabber-Karte* mit dem MTV-Fernsehprogramm gespeist. Durch die ständigen Farbwechsel und die schnellen Bewegungen in den Musikvideos konnte die Beispielapplikation über Stunden auf ihre Stabilität überprüft werden (siehe Bild "MTV Langzeit-Test").

Abbildung 5-12. Tracing-Testframes

Tests



[Zurück](#)
Algorithmen

[Zum Anfang](#)
[Nach oben](#)

[Nach vorne](#)
Weiterentwicklung und
Verbesserung

Weiterentwicklung und Verbesserung

Ein interessanter Ansatz für eine Weiterentwicklung der Beispielapplikation liegt wahrscheinlich in der Verwendung eines adaptiven Thresholding wie es in Kapitel *Segmentierung* Unterpunkt *Adaptives Thresholding* vorgestellt wurde. Hierdurch würden sich ungleichmässige Lichtverhältnisse ausgleichen und das System wäre flexibler in der Wahl des Einsatzortes.

Einen ähnlichen Flexibilitäts-Vorteil erhält man durch eine automatische Bestimmung des Threshold, was in Kapitel *Segmentierung* unter Unterpunkt *Automatische Schwellwertbestimmung* erläutert wurde. Diese Anpassung würde es dem Programm erlauben, Helligkeitsveränderungen, die die gesamte Rennbahn betreffen, auszugleichen. Dies betrifft auch einen Wechsel von Sonnenlicht auf Kunstlicht, der zu einer Veränderung der Farbwerte der Objekte führen kann.

Bei der Wahl der Algorithmen zur Objekterkennung, wurde darauf geachtet, dass eine kontinuierliche Veränderung der Kamera-Perspektive möglich ist. So könnte ein Kamerasystem dem Objekt nachgeführt werden, um den Beobachtungsraum zu vergrössern. Gerade bei der eingesetzten "Sony EVI D30/31" Kamera, die sich auf einem elektronisch schwenkbarem Stativ befindet, welches über eine serielle Schnittstelle angesprochen werden kann, liegt dieser Einsatz nahe.

Eine andere Art der Optimierung entstünde, wenn die Applikation die Fähigkeit erhalten würde, zu erlernen, in welchen Bereichen Objekte zu erwarten sind. So könnte man beispielsweise bei der Modell-Rennbahn nach einer Einlernphase für jedes Fahrzeug nur die Rennstrecke überwachen.

Ein Punkt, den die Applikation noch nicht berücksichtigt, ist die Weitergabe der ermittelten Daten. Während es für eine Beispielapplikation ausreicht, das erkannte Objekt auf dem Monitor hervorzuheben, ist in der Praxis eine Schnittstelle für einen Informationstransfer notwendig. Es wäre beispielsweise bei der Modell-Rennbahn denkbar, die Positionsangaben der einzelnen Rennwagen in TCP Pakete zu verpacken und über Ethernet oder Internet an den Steuerrechner zu übertragen, damit dieser die Steuerung der Fahrzeuge anpassen kann. Ein solcher Aufbau wurde schon in der Abb. *Möglicher Laboraufbau* der *Einleitung* vorgestellt.

Eine andere Frage ist, ob man schon auf dem Rechner, der für die Objekterkennung zuständig ist, die Daten von einem Pixel-Format in ein Meter-Format konvertiert. So liesse sich beispielsweise die Geschwindigkeit und ein Richtungsvektor der einzelnen Rennwagen errechnen, welche an den Steuerrechner übertragen werden könnte. Auf der anderen Seite wäre es auch denkbar, die Steueraufgabe und die Aufgabe der Objekterkennung/Verfolgung direkt auf einem Rechner laufen zu lassen. Hierfür ist zu berücksichtigen, dass beide Applikationen unter dem selben Betriebssystem funktionieren.

Es kann vorkommen, dass sich ein verfolgtes Objekt z.B. eine Modell-Rennbahn von einem besser ausgeleuchteten Bereich in einen schlechter ausgeleuchteten Bereiche bewegt und umgekehrt. Um die Verfolgung der Fahrzeuges sicher zu stellen, wird der Thresholdbereich für das Fahrzeug großzügig bemessen. Alternativ könnte man eine stetige Anpassung der Thresholdschwellen vornehmen, welche sich aus der Farbe des gerade verfolgten Objektes ermitteln ließe. Das gleiche Prinzip könnte auch für die Größe eines Objektes verwendet werden.

Weiterentwicklung und Verbesserung

Gerade die wegen ihrer guten Veranschaulichung gewählte Ausgabe in einem GUI sollte in der Praxis nur zu Test- und Demonstrationszwecken verwendet werden. Durch den Verzicht auf eine grafische Darstellung kann in der Beispielapplikation Rechenleistung eingespart werden. In Testdurchläufen erzeugte das GUI bis zu 95 % der Applikations-Laufzeit (siehe Unterkapitel Methoden-Auslastung). Noch wichtiger ist jedoch, dass die Abhängigkeit zum GUI aufgehoben wird. Ein stark beanspruchtes GUI ist in der Lage, die Applikation in einem fast beliebigen Maß auszubremsen.

Bei der Implementierung des Subframe Algorithmus wird nach einem Objektverlust mit der nächsten Bildabtastung sofort das gesamte Bild nach dem Objekt durchsucht. Ein Optimierungsansatz wäre, die Richtung, in die sich das Objekt zuletzt bewegt hat, weiter zu verfolgen und den Subframe in Stufen zu vergrößern. So wird Rechenzeit eingespart, falls das Objekt z.B. nur kurzzeitig durch einen anderen Körper verdeckt wurde. Als Nachteil kann es länger dauern bis das Objekt wiedergefunden wird.

Durch Verwenden von zwei Kameras, die in einem festen Abstand zueinander das selbe Objekt beobachten, könnte die Position des Objektes im dreidimensionalen Raum bestimmt werden. Eine Technik dies zu realisieren, wäre, das Objekt erst mit einer Kamera zu erfassen und die Informationen wie Farbe und Größe gezielt dazu zu verwenden, es im Bild der anderen Kamera zu suchen. Anhand des Positionsunterschiedes der Objekt-Abbildungen in den beiden Aufnahmen, lässt sich die Entfernung des Objektes zur Kamera ermitteln. Da sich das Objekt, im Bild der zweiten Kamera, auf der selben Höhe befinden muss und nur auf der Achse, in der die Kameras zueinander montiert sind, verschiebt, reicht es aus, einen horizontalen Streifen des Bildes, der ein wenig breiter als die Abmessungen des Objektes ist, zu untersuchen. Die anderen beiden Dimensionen des Raumes ergeben sich direkt aus der Position der Objekte im Bild.

Zurück

Tests

Zum Anfang

Nach oben

Nach vorne

Zusammenfassung und Ausblick

Kapitel 6. Zusammenfassung und Ausblick

Das Ziel der Arbeit war es festzustellen, ob eine "einfache" Kameragestützte Echtzeit Objektverfolgung von beispielsweise der Rennwagen auf der Rennbahn, unter Linux möglich ist. Einfach bedeutet, ohne besonderen Hardwareaufwand. So soll ein 800 MHz PC, handelsübliche Kameras und eine unter dem Betriebssystem Linux lauffähige Software verwendet werden.

Hierfür musste zunächst festgestellt werden, ob die verfügbare Rechenleistung für die Lösung der Aufgabe ausreichend ist. Dazu wurden verschiedene Algorithmen entwickelt und getestet.

Begonnen wird in dieser Arbeit mit der Beschreibung eines Versuchsaufbaues und einer Beispielapplikation zur Kameragestützten Echtzeit. Gerade bei der Beispielapplikation sowie in den beispielprogrammorientierten Abschnitten dieser Arbeit wird speziell auf einer Implementierung unter Linux in der Programmiersprache C++ eingegangen. Es wird eine Technik vorgestellt, mit der Bildinformation von verschiedenen Eingabegeräten (Video-Kamera / USB-Webcam) möglichst effizient in den Arbeitsspeicher übertragen werden kann.

Um die erhaltene Bildinformation in wichtige und unwichtige zu unterteilen, werden verschiedene Techniken der Segmentierung erläutert. darunter Methoden wie die Kantenerkennung, statisches und dynamisches Thresholding sowie das Verfahren der Differenzbildung, welches sich für bewegte Objekte besonders eignet.

Weiter wird erläutert, wie in dem segmentierten Bild die Erkennung von zusammenhängenden Flächen realisiert werden kann. Es wird veranschaulicht, wie das Bild in ein "flächenorientiertes Bild" überführt wird und wie Zusatzinformationen zu den gefundenen Flächen in einer speziellen Tabelle abgelegt werden. Hierbei wird beachtet, dass die Anzahl der Zugriffe gering und die Art des Zugriffes möglichst schnell erfolgt.

Anhand der durch die Segmentierung gesammelten Zusatzinformationen, werden Methoden zur Objektbewertung vorgestellt. Es wird erklärt, wie mit Hilfe definierter Kriterien die Auswahl einer der erkannten Flächen erfolgen kann. Hierbei wird beispielsweise das Verfahren der Kreisflächenähnlichkeit dargestellt und mit der Technik der Mengendichte verglichen. Es erfolgt die Beschreibung eines an die Kreisflächenähnlichkeit angelehnten vereinfachten Verfahrens, welches erheblich weniger Rechenzeit beansprucht.

Durch das Verfolgen der zuvor selektierten Fläche mit einem Fenster, das kleiner als das Gesamtbild ist, wird eine sehr effektive Technik präsentiert, um den Rechenleistungs-Bedarf für die komplette Objektverfolgung zu senken. Weiter wird erläutert, wie es möglich ist, mehr als nur ein Objekt in einem Bild-Strom zu verfolgen und wo die Risiken bei der Verwendung von "Subframing" liegen.

Für die Ausgabe der verfolgten Objekte in Kontext zum eingefangenen Bild wird ein Werkzeug zur Visualisierung vorgestellt. Dieses grafische Fenster ist in der Lage, verschiedene, sich im Speicher befindende Bildinformationen, in den einzelnen Programmentwicklungsabschnitte visuell wiederzugeben. Für die nötigen Kenntnisse, die für die Realisierung eines solchen Fensters interessant sind, geht ein Kapitel auf die X-Window Architektur und speziell auf die Xlib-Bibliothek ein.

Zusammenfassung und Ausblick

Für die Erläuterung spezieller Fachbegriffe, verfügt diese Arbeit über ein Glossar, dass mit einer genaueren Beschreibung der Begriffe sowie mit interessanten Eckdaten dient.

Als Ergebnis der Arbeit kann gesagt werden: *Es ist prinzipiell möglich mit einer Kameragestützte Echtzeit Objekverfolgung unter Linux Objekte wie die Rennwagen einer Modell-Rennbahn zu verfolgen. Dies wurde in dieser Arbeit prototypisch vorgemacht.*

Um möglichst früh zu erkennen, ob eine Kameragestützte Echtzeit Objekverfolgung unter Linux realisierbar ist, wurde für die Entwicklung des Verfahrens des "fast prototyping" verwendet. Im Laufe der Arbeit ließ sich erkennen, dass eine Umsetzung der Aufgabenstellung technisch möglich ist. Ein Problem war jedoch, dass der PC auf dem die Applikation lief, mit 800 MHz für diese Aufgabe nicht performant genug sein könnte. Als Lösung für dieses Problem stellte sich die Technik des "Subframing" heraus. Durch die starke Einsparung an Rechenleistung wurde es möglich beispielsweise drei Objekte gleichzeitig zu verfolgen. So wurden in diesem Punkt, die an die Applikation gestellten Anforderungen übertroffen, ein Objekt verfolgen zu können.

Ein Punkt, der nicht gelöst wurde, ist die Verfolgung der Rennwagen auf der kompletten Rennstrecke. Grund dafür ist, dass die Deckenhöhe des Labors, in dem sich die Modell-Rennbahn befindet ca. einen Meter zu niedrig ist, um mit der Kamera die gesamte Strecke erfassen zu können. Durch eine stark azentrische Position der Kamera über der Strecke, ließ sich zwar die komplette Fahrbahn überwachen, jedoch führte diese Technik zu so starken Bildverzerrungen, dass das Auto am entferntesten Punkt zur Kamera winzig dargestellt wurde und unmittelbar unter der Kamera goßzügig zu erkennen war.

Um die Funktion der Beispielapplikation dennoch anhand der Rennbahn überprüfen zu können, wurde einfach ein Ausschnitt, der ca. 70 % der Rennstrecke beinhaltet, beobachtet. Alternativ wäre es auch möglich, den Öffnungswinkel der Kamera durch eine Linse zu erweitern, was jedoch zu einer (leichten) Bildverzerrung führt, oder direkt eine Kamera einzusetzen, die einen größeren Öffnungswinkel aufweist. Eine weitere Alternative wäre es, einen Spiegel über der Rennstrecke anzubringen, um von unten über den Spiegel die Rennbahn beobachten zu können. Hierdurch würde sich die Länge der optischen Achse über die Deckenhöhe hinaus erweitern lassen. Eine dritte Technik ist, mit zwei oder mehreren Kameras je einen Teil der Strecke zu überwachen, so dass sich mit den Teilbildern wieder die gesamte Strecke beobachten lässt. Bei der Verknüpfung der Teilbilder müssten dann ggf. noch kleine Teilverzerrungen zwischen den Bildern ausgeglichen werden.

Auf den 70 % der Rennstrecke war die Verfolgung von zwei Rennwagen, jeder auf seiner Spur, ohne Zusatzmittel, möglich. Voraussetzung dafür war, dass der Wagen gleichmäßig genug gefärbt war und die Fahrzeugfarbe sich vom Hintergrund abhob. Wurden die Wagen auf einer Fahrzeugfläche von ca. 70% mit einem farbigen Stück Papier beklebt, konnte der Aufbau noch erheblich unempfindlicher gegen Änderungen der äußeren Lichtverhältnisse gemacht werden (siehe Verfolgung der Rennwagen auf der Modell-Rennbahn). Dieser Effekt ist dadurch zu begründen, dass das grob, senkrecht zur optischen Achse der Kamera aufgeklebte, matte Papier das Licht wesentlich gleichmäßiger in die Kamera-Richtung reflektiert, als die glänzende Lackierung der Rennwagen. Unter Verwendung der Papierstücke und einer Kamera mit größerem Öffnungswinkel (wie z.B. die "ToUCam Pro" von Philips) sollte eine Dauerüberwachung der Rennstrecke recht unproblematisch sein.

Abbildung 6-1. Verfolgung der Rennwagen auf der Modell-Rennbahn



Soll die Objektverfolgung real zur Überwachung der Rennwagen eingesetzt werden, ist es notwendig, sie in folgenden Punkten zu erweitern:

- Es muss eine Kommunikations-Schnittstelle geschaffen werden, mit der die Programme, die für die Steuerung der Rennwagen zuständig sind, mit der Objektverfolgung kommunizieren können. Neben der Abfrage der Position ist es auch nötig, z.B. Kriterien über das zu verfolgende Objekt übergeben zu können.
- Das Problem, das zur Zeit nur ein Teil der Bahn überwacht wird, ist zu lösen.
- Verzerrungen, die durch die Optik der Kamera oder den Betrachtungswinkel der Kamera entstehen, müssen ausgeglichen werden.
- Die Objekterkennung muss so parametrisiert werden, dass die Objekte sicher verfolgt werden. Die beiden Fahrzeuge sollten für optimale Ergebnisse großflächig, gleichmässig und je in einer Farbe, die sich von der Umgebung und dem anderen Fahrzeug abhebt, lackiert bzw. kenntlich gemacht werden.
- Die Kanalwahl bei Verwendung einer Frame-Grabber-Karte sollte innerhalb der Applikation erfolgen und nicht über ein zusätzliches Tool.
- Für bessere Ergebnisse könnten die Optimierungen aus dem Unterkapitel Weiterentwicklung und Verbesserung des Kapitels Diskussion und Bewertung umgesetzt werden.

Denkbare Einsatzgebiete für eine solche Objektverfolgung wären neben einer Positionserkennung von Rennwagen auf einer Modell-Rennbahn, z.B. das Beobachten des Weges einer Labormaus durch ein Labyrinth, das Absichern eines Gegenstandes gegen Diebstahl, das Überprüfen der Flussrichtung von Teilchen einer gewissen Größe in einem Medium oder das Erfassen der Position eines Roboters in einem Raum, um ihn um Hindernisse und zu gewissen Punkten zu steuern.

[Zurück](#)

Weiterentwicklung und
Verbesserung

[Zum Anfang](#)

[Nach vorne](#)

Konfiguration des Systems

Anhang A. Konfiguration des Systems

Die Beispielapplikation wurde unter dem freien Betriebssystem Linux entwickelt. Als Distribution diente Debain in der Version 3.1 "sid".

Für die Verwendung einer Frame-Grabber-Karte ist es erforderlich, den Kernel des Linux Systems mit Video4Linux Support zu kompilieren. Bei der in der Diplomarbeit verwendeten Frame-Grabber-Karte wurden folgende Optionen bei der Erstellung des Kernels zusätzlich aktiviert:

- Loadable module support --->
 - ◆ [*] Enable loadable module support
 - ◆ [*] Set version information on all module symbols
 - ◆ [*] Kernel module loader
- Multimedia devices --->
 - ◆ (M) Video For Linux
 - ◆ Video For Linux --->
 - ◇ [*] V4L information in proc filesystem
 - ◇ (M) I2C on parallel port
 - ◇ --- Video Adapters
 - ◇ (M) BT848 Video For Linux
 - ◇ ((M) SAA5249 Teletext processor)
- Character devices --->
 - ◆ ((M) Enhanced Real Time Clock Support)

Die Punkte in der Sparte "Loadable module support" sind nur nötig, da einige Funktionalitäten als Modul erzeugt werden. Die entsprechenden Einträge sind in der vorangehenden Auflistung an einem "(M)" zu erkennen. Der Vorteil von Modulen ist, dass sie vom Kernel bei Bedarf nachgeladen oder entladen werden können. So belegen sie nicht ständig Speicherplatz und der Kernel bleibt kleiner. Weiter ist es möglich, ein spezielles Modul zu aktualisieren oder zu verändern (engl. patchen), ohne den kompletten Kernel neu erzeugen zu müssen.

Der letzte Punkt *Enhanced Real Time Clock Support* bewirkt, dass bei der Zeiterfassung für einen Grabvorgang präzisere Ergebnisse ausgegeben werden. Als Kernel wurde die Version 2.4.14 und 2.4.18 verwendet.

Da die Beispielapplikation eine grafische Oberfläche verwendet, muss auf dem System ein X-Windows System installiert werden. Auf dem Entwicklungssystem wurde XFree86 - X11R6 als X-Server und KDE 2.2.1 als Oberfläche eingesetzt.

Konfiguration des Systems

Um die USB-Webcam "TerraCAM USB Pro" von TerraTec verwenden zu können, war es notwendig, den Kernel mit USB Support für die Kamera zu kompilieren. Da wie bei der Frame-Grabber-Karte Video4Linux verwendet wird, entsprechen sich die Konfiguration im ersten Bereich. Erst mit "USB support --->" werden die USB spezifischen Einstellungen vorgenommen. In der Sektion "--- USB Controllers" reicht es aus die der drei Optionen zu wählen, die zu dem entsprechenden (Mainboard) Chipsatz passt. Da der Treiber als Modul geladen wird, ist es notwendig, das entsprechende Modul mit dem Befehl "modprobe usb-uhci" und "modprobe ov511" zu laden. Alternativ kann "usb-uhci" und "ov511" in die Datei "/etc/modules" eingetragen werden, um die Module bei jedem Systemstart zu laden.

- Loadable module support --->
 - ◆ [*] Enable loadable module support
 - ◆ [*] Set version information on all module symbols
 - ◆ [*] Kernel module loader
- Multimedia devices --->
 - ◆ (M) Video For Linux
 - ◆ Video For Linux --->
 - ◇ [*] V4L information in proc filesystem
 - ◇ (M) I2C on parallel port
- Character devices --->
 - ◆ ((M) Enhanced Real Time Clock Support)
- USB support --->
 - ◆ (*) Preliminary USB device filesystem
 - ◆ --- Miscellaneous USB options
 - ◆ --- USB Controllers
 - ◆ (M) UHCI (Intel PIIX4, VIA, ...) support
 - ◆ (M) UHCI Alternate Driver (JE) support
 - ◆ (M) OHCI (Compaq, iMacs, OPTi, SiS, ALi, ...) support
 - ◆ --- USB Multimedia devices
 - ◆ (M) USB OV511 Camera support

In dem für die Beispielapplikation verwendeten Computersystem wurde ein AMD Athlon mit 800 MHz und 384 MB Arbeitsspeicher verwendet. Die Belastung des System durch die Applikation lag bei ca. 20-50 %. Der Arbeitsspeicher wurde nur zu einem geringen Bruchteil durch die Anwendung benutzt. Bei besonders ungünstigen Objekten und Threshold-Filtern kann es jedoch vorkommen, dass die Systemrechenleistung nicht ausreicht und Bilder verloren gehen. Für das Grabben des Bildes wurde die Frame-Grabber-Karte "WinTV PCI FM" der Firma Haupauge verwendet, auf der der Videodigitizer-Chip Bt878 der Firma Connexant (früher Brooktree) eingesetzt wird. Unter Linux ist es wichtig darauf zu achten, dass die Frame-Grabber-Karte unterstützt wird. Basiert die Karte auf den weit verbreiteten BT848, BT878 oder BT879 Videodigitizer-Chip, sollten sich Komplikationen vermeiden lassen. Bei der Verwendung einer USB-Kamera sollte unbedingt sichergestellt werden, dass die Kamera von Linux unterstützt wird.

Die Ausgabe auf dem Monitor übernahm eine "3D PROPHET II MX DUAL-DISPLAY VIDEO" Grafikkarte mit Nvidia Chipsatz und 32 MB Speicher. Hierbei wurde nicht das für NVIDIA verfügbare binäre Kernel-Modul eingesetzt, noch wurde getestet, wie sich die Verwendung dieses Kernel-Moduls auf die Beispielapplikation auswirkt.

Konfiguration des Systems

Bevor die Beispielapplikation bei Verwendung einer Frame-Grabber-Karte ausgeführt wird, ist es wichtig, dass mit einem TV-Tool wie z.B. xawtv von Gerd Knorr die Eingangsschnittstelle gewählt wird. Bei dem Testaufbau wurde beispielsweise fast immer der S-Video Eingang gewählt. Wird der Tuner der TV Karte verwendet, muss zusätzlich mit Hilfe des TV-Tool eine entsprechende Frequenz gewählt werden, auf der sich die Bildinformation befindet. Ist mittels TV-Tool ein Bild zu erkennen, kann diese Applikation geschlossen werden und mit der Ausführung des Beispielprogrammes begonnen werden. Die Verwendung der Frame-Grabber-Karte durch mehrere Applikationen, zur gleichen Zeit, ist nicht möglich. Dieser Einstellungsvorgang muss nach jedem Neustart des Computersystems wiederholt werden.

Für das Kompilieren der Beispielapplikation wurde der KDevelop Version 2.2.2 verwendet. Um die Verwendung der Xlib Funktionalitäten zu ermöglichen, ist es nötig, unter "Projekt->Optionen->Linker Options" bei "additional Libraries" den Parameter "-L/usr/X11R6/lib" anzugeben. Außerdem muss im selben Fenster unter "libraries" die "X11" Library angewählt werden.

[Zurück](#)

Zusammenfassung und Ausblick

[Zum Anfang](#)

[Nach vorne](#)

Bedienung der Beispielapplikation

Anhang B. Bedienung der Beispielapplikation

Um die Beispielapplikation zu verwenden, ist es notwendig sicherzustellen, dass der Kernel des Systems mit den im Kapitel *Konfiguration des Systems* beschriebenen Optionen kompiliert und dass ggf. die dort erwähnten Module geladen wurden.

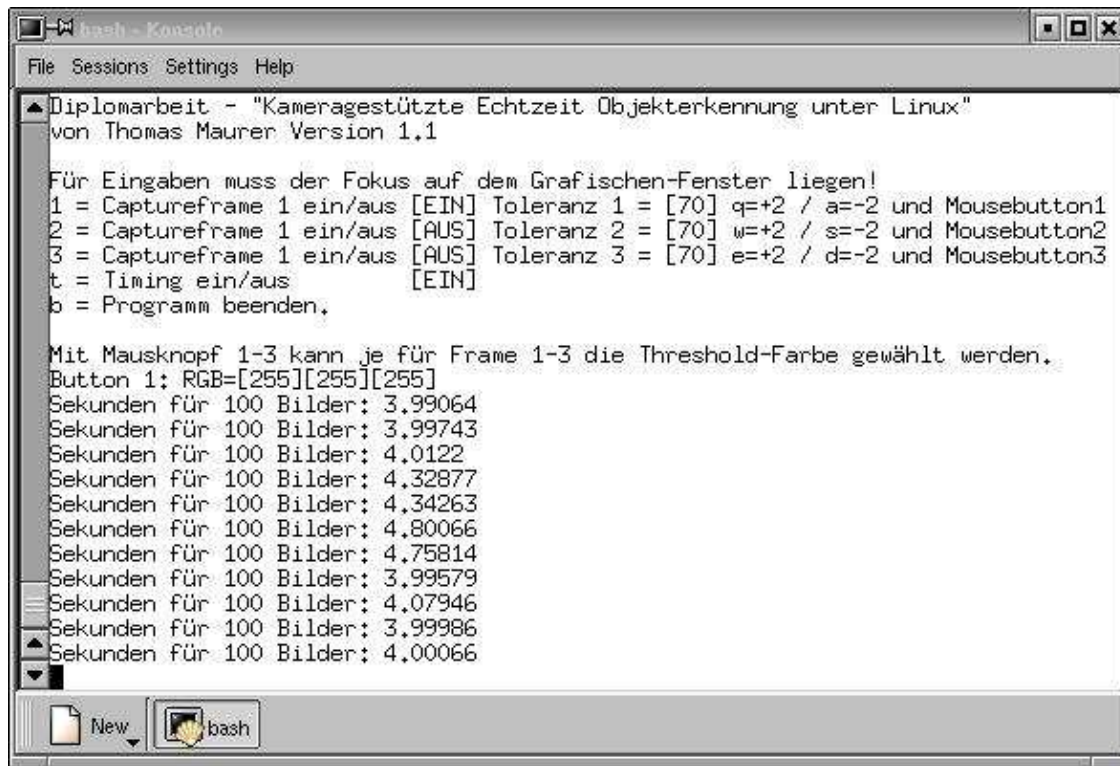
Um die Beispielapplikation zu installieren, reicht es aus, das TAR-Archiv mit "tar -xvzf beispielapplikation.tar.gz neuesverzeichnis/" in ein Verzeichnis zu entpacken. Danach wird in das neue Verzeichnis gewechselt und mit dem Befehl "make" die Applikation erzeugt. Eine Voraussetzung ist, dass der Compiler gcc installiert ist. Wurde die Applikation erzeugt, kann mit "./video" das Programm gestartet werden.

Als Parameter kann der Applikation das Video Device übergeben werden. Ein Aufruf in der Form "./video /dev/video1" würde dazu führen, dass versucht wird, Bilder von Device "/dev/video1" zu lesen. Gerade bei Verwendung einer *USB*-Kamera ist es wichtig, dass die Kamera angeschlossen ist, da ansonsten auf das Device nicht zugegriffen werden kann. Bei einem Start ohne Parameter wird automatisch versucht, dass Device "/dev/video" zu verwenden.

Die Beispielapplikation verfügt über ein einfaches Interface, mit dessen Hilfe sich Einstellungen zur Verfolgung von drei unabhängigen Objekten treffen lassen (siehe Abb. *Aufbau einer Objekterkennung*). Für jede Interaktion wird das Menü neu dargestellt und zeigt die aktuellen Einstellungen an. Da das Menü bei jeder Eingabe erneut in die Konsole geschrieben wird und die alten Informationen/Einstellungen nach oben rücken, ist darauf zu achten, dass nur die am weitesten unten stehenden Einstellungen aktuell sind.

Abbildung B-1. Aufbau einer Objekterkennung

Bedienung der Beispielapplikation



```
bash - Konsole
File Sessions Settings Help

▲Diplomarbeit - "Kameragestützte Echtzeit Objekterkennung unter Linux"
  von Thomas Maurer Version 1.1

Für Eingaben muss der Fokus auf dem Grafischen-Fenster liegen!
1 = Captureframe 1 ein/aus [EIN] Toleranz 1 = [70] q=+2 / a=-2 und Mousebutton1
2 = Captureframe 1 ein/aus [AUS] Toleranz 2 = [70] w=+2 / s=-2 und Mousebutton2
3 = Captureframe 1 ein/aus [AUS] Toleranz 3 = [70] e=+2 / d=-2 und Mousebutton3
t = Timing ein/aus [EIN]
b = Programm beenden.

Mit Mausknopf 1-3 kann je für Frame 1-3 die Threshold-Farbe gewählt werden.
Button 1: RGB=[255][255][255]
Sekunden für 100 Bilder: 3.99064
Sekunden für 100 Bilder: 3.99743
Sekunden für 100 Bilder: 4.0122
Sekunden für 100 Bilder: 4.32877
Sekunden für 100 Bilder: 4.34263
Sekunden für 100 Bilder: 4.80066
Sekunden für 100 Bilder: 4.75814
Sekunden für 100 Bilder: 3.99579
Sekunden für 100 Bilder: 4.07946
Sekunden für 100 Bilder: 3.99986
Sekunden für 100 Bilder: 4.00066
```

Da das Verarbeiten der Maus und die Tastatureingaben über das grafische Fenster (nicht das Konsolen Fenster) erfolgt, ist es wichtig, dass dieses Fenster den Window-Fokus hat. Erscheint beim Drücken einer Taste das entsprechende Zeichen in der Konsole, liegt der Fokus auf der Konsole und das Programm kann nicht bedient werden. Abhilfe schafft z.B. das Klicken auf die Kopfzeile des grafischen Fensters.

Die Tasten "1", "2" und "3" aktivieren und deaktivieren den entsprechenden ersten, zweiten oder dritten Verfolgungsvorgang .

Mit den Tasten unterhalb der "1" also "q" und "a", unterhalb der "2" also "w" und "s" und unterhalb der "3" also "e" und "d" kann die Toleranz des jeweiligen Thresholdwertes verändert werden. Das Drücken von "q" würde beispielsweise den Thresholdwert für das erste Objekt erhöhen und "a" den Wert senken. Damit der Wert auf das Objekt angewendet wird, muss das entsprechende Objekt mit der Maus angeklickt werden. Die Toleranz kann zwischen den Werten 0 und 254 in Zweierschritten variiert werden. Je größer die Toleranz gewählt wird, um so mehr darf die zu erkennende Farbe von der Farbe des angeklickten Pixels abweichen.

Mit der linken, mittleren und rechten Maustaste kann im grafischen Fenster ein Pixel angeklickt werden. Die Farbe für dieses Pixel wird automatisch zusammen mit der eingestellten Toleranz als Thresholdwerte für den jeweiligen Verfolgungsvorgang gewählt. Die linke Maustaste ist dabei der ersten Verfolgung zugeordnet, die mittlere Taste dem zweiten und die rechte Taste dem dritten Verfolgungsvorgang.

Mit der Taste "t" kann die Zeiterfassung ein und ausgeschaltet werden. Ist die Zeiterfassung aktiv, stoppt die Applikation die Zeit, die für das Verarbeiten von 100 Bildern benötigt wird. Dieser Wert wird auf der Konsole ausgegeben.

Durch Drücken eine beliebigen Taste, die mit keiner Programmfunktion belegt ist wie beispielsweise der Leerzeichen-Taste, lässt sich das Menü jederzeit neu darstellen.

Mit der Taste "b" kann die Applikation beendet werden.

Zurück

Konfiguration des Systems

Zum Anfang

Nach vorne

Quellcode der Beispielapplikation

Anhang C. Quellcode der Beispielapplikation

Abbildung C-1. Quellcode der Beispielapplikation

```

/*****
main.cpp - Beispielapplikation zur Diplomarbeit
          Kameragestützte Echtzeit Objektverfolgung unter Linux.

          -----
copyright      : (C) 2002 by Thomas Maurer
email         : thomas@maurer-tech.com
*****/

/*****
*
*   This program is free software; you can redistribute it and/or modify
*   it under the terms of the GNU General Public License as published by
*   the Free Software Foundation; either version 2 of the License.
*
*****/

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

// für Xlib
#include <X11/Xlib.h>
#include <stdlib.h>

// #include <curses.h>
#include <iostream.h>
#include <stdio.h>
#include <sys/time.h>
#include <fcntl.h>
#include <limits.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <linux/videodev.h>

#define VIDEO_BUFFERS    2 // Es werden 2 Buffer verwendet
#define debug    0
// 0 = aus
// 1 = labels Tabelle überwachen
// 3 = Grenzen des geklickten Punkes in grafischen Ausgabefenster

// Bild Eigenschaften festlegen
#define WIDTH    320 // 640 // 320 // 924 // 320 // 80

```

Quellcode der Beispielapplikation

```
#define HEIGHT 240 // 480 // 240 // 576 // 240 // 60
#define DEPTH 3 // 3 = 3 byte = VIDEO_PALETTE_RGB24 = RGB888 in 24bit Worten.
#define IMG_SIZE (WIDTH * HEIGHT * DEPTH)
#define GRAY_STEP 1 // Gibt an, wie groß der Helligkeitsunterschied
// zwischen 2 Flächen sein soll. Nur für Testzwecke! Sonst 1
// #define ARRAY_ZEILEN (WIDTH*HEIGHT/4)
#define ARRAY_ZEILEN (WIDTH*HEIGHT*GRAY_STEP/4+1)
#define ARRAY_SPALTEN 9 // 0-8 = 9
#define NEXT_INDEX 0
#define MENGE 1
#define X_MAX 2
#define X_MIN 3
#define Y_MAX 4
#define Y_MIN 5
#define MENGENDICHTE 6
#define CONNECTIONS 7
#define ROOT 8

/**
 * Diese Klasse enthält Methoden für die Findung und Verfolgung von Objekten.
 */
class o_tracing
{
public:
    o_tracing();
    ~o_tracing();
    unsigned int * areas_buffer;
    void start_tracing(unsigned char * in_org_frame);
    void set_rgb_threshold(unsigned char red_min, unsigned char red_max,
        unsigned char green_min, unsigned char green_max,
        unsigned char blue_min, unsigned char blue_max);
    void set_total_threshold(unsigned int distance);

    void draw_tracing_frame();
    void draw_tracing_frame(unsigned char * org_frame);
    unsigned int create_roots(unsigned int index, unsigned int root);
private:
    unsigned int id;
    unsigned char * org_frame;
    // Tabelle die für das eindeutige Labeling verwendet wird.
    unsigned int labels[ARRAY_ZEILEN][ARRAY_SPALTEN];
    // unsigned int akt_label;
    unsigned int x0;
    unsigned int y0;
    unsigned int width;
    unsigned int height;
    unsigned int akt_label; // Maximale Anzahl der labels
    unsigned int objekt_index; // Index des gefundenen Objektes
    bool rgb_tracing;
    unsigned int distance;
    unsigned char red_min;
    unsigned char red_max;
    unsigned char green_min;
    unsigned char green_max;
    unsigned char blue_min;
    unsigned char blue_max;
    unsigned int old_x_max, old_x_min, old_y_max, old_y_min, old_xcenter, old_ycenter;

    void threshold();
    void tracing();
    void labeling(unsigned int area_index);
    void just_count(unsigned int index1, unsigned int modulo, unsigned int quotient, unsigned int
```

Quellcode der Beispielapplikation

```
void join_labels();
void in_label(unsigned int index1, unsigned int index2,
              unsigned int modulo, unsigned int quotient);
unsigned int get_mengendichte(unsigned int index);
};

/**
 * Diese Klasse enthält Methoden, die für das Einfangen von Bildern
 * von einem Video Device zuständig sind.
 */
class video_in{

public:
    video_in();
    int grab_open (char *device, int width, int height, int depth);
    void grab_close (void);
    int grab_frame (int frame);
    unsigned char * grab_pix (void);
private:
    unsigned char *buffer;
    int video_fd;          // Video filedescriptor
    int frame;

    struct video_mbuf mbuf;
    struct video_picture pic;
    struct video_channel chn;
    struct video_mmap buf[VIDEO_BUFFERS]; // 2 Buffer, die das gegrabte Bild enthalten.
    struct video_capability cap;
    int grab_frame ();
    int grab_sync ();
};

/**
 * Diese Klasse enthält Methoden, die für das Ausgeben von Bildern
 * auf einem X-Window-System und zur Verarbeitung von Ereignissen verwendet werden.
 */
class video_out{

public:
    video_out();
    int get_depth();
    bool get_event(unsigned char *buffer, unsigned char *type,
                  unsigned char *red, unsigned char *green, unsigned char *blue, KeySym *key);
    void display_frame (unsigned char * translated_buffer, int depth, int bpl,
                      unsigned int *buffer);
    void display_frame (unsigned char * translated_buffer, int depth, int bpl,
                      unsigned char *buffer);
private:
    Window create_simple_window(Display* display, int width, int height, int x, int y);
    Display* display; // Zeiger auf X Display Struktur
    int screen_num; // Nummer des Screens
    Window win; // Zeiger zum Fenster der Applikation
    char *display_name;
    GC gc; // GC (graphics context) wird zum Zeichnen benötigt
    XGCValues gcv; // Enthält Einstellungen für den GC
    XEvent an_event; // Variable für ein Ereigniss
    XImage *ximage; // XImage das im Fenster angezeigt werden soll
    void handle_button_down(XButtonEvent* button_event, unsigned char *buffer,
                          unsigned char *type, unsigned char *red, unsigned char *green, unsigned char *blue);
};
```

Quellcode der Beispielapplikation

```
/**
 * Der Konstruktor der Klasse o_tracing initialisiert einige Variablen
 */
o_tracing::o_tracing(){

    static int id_count=0;
    id_count++;
    id=id_count;
    x0=0;
    y0=0;
    width=WIDTH;
    height=HEIGHT;
    akt_label=(WIDTH*HEIGHT/4);
    objekt_index=0; // Index des gefundenen Objektes
    // Anlegen eines Speicherbereichs, in dem die gefundenen Flächen abgelegt werden
    areas_buffer=(unsigned int *)malloc(WIDTH*HEIGHT*4); //mal 4 wegen 4 byte integer Werten);

    rgb_tracing=false;
    distance=400;

    red_min=0;
    red_max=0;
    green_min=0;
    green_max=0;
    blue_min=0;
    blue_max=0;
}

/**
 * Der Destruktor der Klasse o_tracing gibt einen
 * Speicherbereich wieder frei.
 */
o_tracing::~o_tracing(){
    free((void *)areas_buffer);
}

/**
 * Diese Funktion durchläuft beginnend bei ihrem Einstiegspunkt
 * rekrusiv den labels Teilbaum und erzeugt
 * für jeden Index einen Root Eintrag.
 * Dabei ist zu beachten, dass ab dem Einstiegspunkt nur die
 * weiteren Elemente bearbeitet werden, die eine Verbindung zu dem
 * Einstiegspunkt haben. Dadurch muss dieser Alg. auf allen Indexen
 * gestartet werden.
 */
unsigned int o_tracing::create_roots(unsigned int index, unsigned int root){
    // Gibt es schon einen Root?
    if(labels[index][ROOT]==0){
        // Es gibt keinen root
        if(labels[index][NEXT_INDEX]!=0){ // Gibt es einen NEXT_INDEX
            // JA Dann überprüfe diesen.
            root = create_roots(labels[index][NEXT_INDEX], root);
        }
        labels[index][ROOT]=root;
        // Werte auf Root übertragen und beim nichtroot nullen.
        if(index!=root){
            labels[root][MENGE] += labels[index][MENGE];
            labels[index][MENGE]=0;
            labels[root][CONNECTIONS] += labels[index][CONNECTIONS];
            labels[index][CONNECTIONS]=0;
            if(labels[root][X_MAX] < labels[index][X_MAX]){
                labels[root][X_MAX]=labels[index][X_MAX];
            }
        }
    }
}
```

Quellcode der Beispielapplikation

```

    }
    labels[index][X_MAX]=0;
    if(labels[root][X_MIN] > labels[index][X_MIN]){
        labels[root][X_MIN]=labels[index][X_MIN];
    }
    labels[index][X_MIN]=0;
    if(labels[root][Y_MAX] < labels[index][Y_MAX]){
        labels[root][Y_MAX]=labels[index][Y_MAX];
    }
    labels[index][Y_MAX]=0;
    if(labels[root][Y_MIN] > labels[index][Y_MIN]){
        labels[root][Y_MIN]=labels[index][Y_MIN];
    }
    labels[index][Y_MIN]=0;
}

// Es gibt einen Root
else root = labels[index][ROOT];
return(root);
};

/**
 * Diese Funktion durchläuft das gesamte aktive Bild und überprüft
 * die Punkte, die in einer Gruppe sind, das Label des 1. Gruppenmitglieds
 * bekommen. Weiter werden die min und max Werte sowie der Mengen
 * Zähler auf das 1. Gruppenelement übertragen.
 * Zuvor werden die roots gebildet.
 * Der Eintrag mit dem grössten connections Wert wird in objekt_index
 * abgelegt und repräsentiert das zu verfolgende Objekt.
 */
void o_tracing::join_labels(){
    unsigned int area_index;
    unsigned int i=0;
    unsigned int tmp=GRAY_STEP; // Compilerbug überlisten.
    // unsigned int mengendichte=0;
    unsigned int connections=0;
    unsigned int tmp2=0;
    unsigned int md_index=0; // Index mit der höchsten Mengendichte

    // Durchlaufe die Labels Tabelle, um alle ROOTs zu bilden und die Mengendichten
    // zu bilden und die größte Mengendichte zu ermitteln.
    for(unsigned int i2=tmp; i2<=akt_label; i2=i2+tmp){
        create_roots(i2, i2);
        // Der auskommentierte Bereich basiert noch auf
        // der Technik der MENGENDICHTE. Diese wurde
        // durch die CONNECTIONS ersetzt.
        /*
        if (labels[i2][MENGE]!=0){
            tmp2 = get_mengendichte(i2);
            labels[i2][MENGENDICHTE]=tmp2;
            if (tmp2 > mengendichte){
                mengendichte = tmp2;
                md_index = i2;
            }
        }
        */
        if (labels[i2][MENGE]!=0){
            tmp2 = labels[i2][CONNECTIONS];
            if (tmp2 > connections){
                connections = tmp2;
                md_index = i2;
            }
        }
    }
}

```

Quellcode der Beispielapplikation

```

    }

    if (md_index!=0){
        area_index = labels[md_index][X_MIN] + (labels[md_index][Y_MIN]*(WIDTH)); // Index im Fläc
        // Durchlaufe den SubFrame im Fenster
        do{
            if((areas_buffer[area_index])!=0){
                if (labels[areas_buffer[area_index]][ROOT]==md_index){
                    areas_buffer[area_index] = 255;
                }
                else areas_buffer[area_index] = 0;
            }
            if(i < width-1){ // Nächstes
                i = i++;
                area_index = area_index + 1;
            } else{
                i = 0;
                // Berich der nicht zum Subframe gehört überstpringen
                area_index = area_index + (WIDTH-width+1);
            }
        }while(area_index <= labels[md_index][X_MAX] + WIDTH*(labels[md_index][Y_MAX]));
    }
    objekt_index=md_index;
#ifdef debug==1
    cout << "2. Durchlauf\n";
    for(unsigned int i=0;i<=akt_label;i++){
        cout << i << "\t";
        cout << labels[i][0] << "\t";
        cout << labels[i][1] << "\t";
        cout << labels[i][2] << "\t";
        cout << labels[i][3] << "\t";
        cout << labels[i][4] << "\t";
        cout << labels[i][5] << "\t";
        cout << labels[i][6] << "\t";
        cout << labels[i][7] << "\t";
        cout << labels[i][8] << "\n";
    }
    cout << "----\n";
#endif
}

/**
 * Für den angegebenen index werden die Zählerstände erhöht
 * und die min max Werte aktualisiert.
 */
void o_tracing::just_count(unsigned int index1, unsigned int modulo, unsigned int quotient, unsig
    if(labels[index1][X_MAX]< modulo){
        labels[index1][X_MAX]=modulo;
    }
    if( (labels[index1][X_MIN]> modulo) || (labels[index1][MENGE] == 0) ){
        labels[index1][X_MIN]=modulo;
    }
    if(labels[index1][Y_MAX]< quotient){
        labels[index1][Y_MAX]=quotient;
    }
    if( (labels[index1][Y_MIN]> quotient) || (labels[index1][MENGE] == 0) ){
        labels[index1][Y_MIN]=quotient;
    }
    labels[index1][MENGE]=labels[index1][MENGE]+1;
    labels[index1][CONNECTIONS]=labels[index1][CONNECTIONS]+connetions;
}

```


Quellcode der Beispielapplikation

```
/**
 * Diese Funktion erstellt eine Tabelle die für jedes Label die Anzahl
 * der Pixel und deren äusterten Positionen in den X-Y System umfasst.
 * In in der Spalte 'index' wird eine Verbindung zu 'wert' geschaffen.
 * Dadurch wird gezeigt, dass index und wert zu einer Gruppe gehören.
 */
void o_tracing::in_label(unsigned int index1, unsigned int index2,
    unsigned int modulo, unsigned int quotient){
    if(index1>index2){
        unsigned int tmp;
        tmp=index1;
        index1=index2;
        index2=tmp;
        /*DEBUG01*/// cout << 'x';
    }
    if(labels[index1][NEXT_INDEX]==0){
        // Diesem index ist noch keine weitere Gruppe zugeordnet.
        labels[index1][NEXT_INDEX]=index2;
    } else{
        // Es sind schon mehr als 2 Gruppen verbunden.
        if(labels[index1][NEXT_INDEX]==index2){// just_count(labels, index1, modulo, quotient);
        else in_label(labels[index1][NEXT_INDEX], index2, modulo, quotient);
    }
}

/**
 * Diese Methode implementiert die Flächenerkennung nach dem
 * 'Connected Components Labeling Algorithmus' Verfahren.
 * Vordergrundpixel die aneinander angrenzen werden zu einer
 * Fläche verbunden.
 */
void o_tracing::labeling (unsigned int area_index){

    unsigned int tmp=0; // Hilfsvariable
    unsigned int pos1=0; // Pixel linkes
    unsigned int pos2=0; // Pixel oben
    unsigned int pos3=0; // Pixel rechts oben
    unsigned int pos4=0; // Pixel links oben
    bool left=true;
    bool up=true;
    bool right=true;
    unsigned int modulo=0;
    unsigned int quotient =0;

    modulo = area_index%WIDTH; // Modulo berechnen um die X-Positon zu ermitteln.
    quotient = area_index/WIDTH; // Division um die Y-Positon zu ermitteln

    // Es muß auf 3 Sonderfälle überprüft werden, damit nicht Information außerhalb
    // des Frames gelesen werden.
    // 1. Der Punkt befindet sich NICHT am linken Rand des Frame
    if(modulo != x0) left=false;
    // 2. Der Punkt befindet sich NICHT am oberen Rand des Frame
    if(area_index >= WIDTH) up=false; // >= da ab Null gezählt wird
    // 3. Der Punkt befindet sich NICHT am rechten Rand des Frame
    if(modulo != x0+width-1) right=false;

    // linkes Pixel lesen
    if(left == false){
        pos1 = areas_buffer[area_index-1];
        // Pixel links oben lesen
        if(up == false){
            pos4 = areas_buffer[area_index-(WIDTH+1)];

```

Quellcode der Beispielapplikation

```
    }
}
// Pixel oben lesen
if(up == false){
    pos2 = areas_buffer[area_index-(WIDTH)];
    // Pixel rechts oben lesen
    if(right == false){
        pos3 = areas_buffer[area_index-(WIDTH-1)];
    }
}
// pos1 + pos2 + pos3 + pos4 = 0 Das Pixel bekommt ein neue Label
if (pos1 + pos2 + pos3 + pos4 == 0){
    (akt_label)=(akt_label)+GRAY_STEP;
    // die Anzahl und die min+max Positionen werden abgelegt
    just_count(akt_label, modulo, quotient, 0);
    areas_buffer[area_index]=akt_label;
    return;
}
// 1 xor 2 xor 3 xor 4 == true (es gibt genau einen Nachbarn)
if( (pos1?1:0) + (pos2?1:0) + (pos3?1:0) + (pos4?1:0) == 1){
    // die Anzahl und die min+max Positionen werden abgelegt
    just_count(akt_label, modulo, quotient, 1);
    // Es ist nur 1 vor den 4 != 0. Dieses wird als Label verwendet.
    areas_buffer[area_index]=pos1 + pos2 + pos3 + pos4;
    return;
}
// Wenn der Programmfluss bis hierher kommt, gibt es min. 2 Labels die das Pixel
// beeinflussen.
// Für das weitere Vorgehen müssen pos1-4 sortiert werden (pos1=min. pos4=max.)
// Dafür wird ein statischer bubble-sort Alg. verwendet.
if(pos1>pos2){
    tmp=pos2;
    pos2=pos1;
    pos1=tmp;
}
if(pos2>pos3){
    tmp=pos3;
    pos3=pos2;
    pos2=tmp;
}
if(pos3>pos4){
    tmp=pos4;
    pos4=pos3;
    pos3=tmp;
}
if(pos1>pos2){
    tmp=pos2;
    pos2=pos1;
    pos1=tmp;
}
if(pos2>pos3){
    tmp=pos3;
    pos3=pos2;
    pos2=tmp;
}
if(pos1>pos2){
    tmp=pos2;
    pos2=pos1;
    pos1=tmp;
}
// Sind Elemente von pos1-pos4 gleich, werden sie in die labels_tabelle
//eingetragen.
```

Quellcode der Beispielapplikation

```
if(pos1){
    areas_buffer[area_index]=pos1;
    if(pos1!=pos2){
        in_label(pos1, pos2, modulo, quotient);
    }
    if(pos2!=pos3){
        in_label(pos2, pos3, modulo, quotient);
    }
    if(pos3!=pos4){
        in_label(pos3, pos4, modulo, quotient);
    }
    just_count(pos1, modulo, quotient, 4);
} else if(pos2){
    areas_buffer[area_index]=pos2;
    if(pos2!=pos3){
        in_label(pos2, pos3, modulo, quotient);
    }
    if(pos3!=pos4){
        in_label(pos3, pos4, modulo, quotient);
    }
    just_count(pos2, modulo, quotient, 3);
} else if (pos3){
    areas_buffer[area_index]=pos3;
    if(pos3!=pos4){
        in_label(pos3, pos4, modulo, quotient);
    }
    just_count(pos3, modulo, quotient, 2);
}
//pos4 gibt es nicht da 1 von 4 schon in einer früheren
// Regel abgefangen wurde
return;
}

/**
 * Diese Methode führt die Aufgabe des Thresholding durch.
 * In ihr wird also entschieden, ob ein Pixel Vordergrund oder
 * Hintergrundpixel wird.
 * Für ein Vordergrundpixel wird sofort das Labeling initiiert.
 */
void o_tracing::threshold(){
    //Löschen der labels Tabelle;
    for(unsigned int i=0;i<=akt_label;i++){
        labels[i][NEXT_INDEX]=0;
        labels[i][MENGE]=0;
        labels[i][X_MIN]=WIDTH;
        labels[i][X_MAX]=0;
        labels[i][Y_MIN]=HEIGHT;
        labels[i][Y_MAX]=0;
        labels[i][MENGENDICHTE]=0;
        labels[i][CONNECTIONS]=0;
        labels[i][ROOT]=0;
    }
    // Thresholding
    akt_label=0;    // Aktuelle Label-Nummer
    // Index des Originals und des Area auf linke obere Ecke des SubFrame stellen.
    unsigned int org_index = (x0 + (y0*(WIDTH)))*3; // Index im gegrabte Framebereich.
    unsigned int area_index = x0 + (y0*(WIDTH));    // Index im Flächenbereich.
    unsigned int i=0;
    unsigned int tmp;
    // Durchlaufe den SubFrame im Fenster
    do{
        // Test ob es sich um ein Vordergrund-Pixel handelt.
```

Quellcode der Beispielapplikation

```
if (rgb_tracing==true){ // Thresholding mittels drei Wertebereiche
    // für RGB bzw. wirklich B G R
    if ( (red_min <= org_frame[org_index+2] && org_frame[org_index+2] <= red_max)
        && (green_min <= org_frame[org_index+1] && org_frame[org_index+1] <= green_max)
        && (blue_min <= org_frame[org_index] && org_frame[org_index]<= blue_max) )
    {
        // Es ist ein Vordergrund-Pixel
        // 'Connected Components Labeling Algorithmus' durchführen
        labeling(area_index);
    } else{
        // Es ist ein Hintergrund-Pixel
        areas_buffer[area_index]=0;
    }
} else { //Thresholding mittels einer Distanz
    tmp = org_frame[org_index+0];
    tmp = tmp + org_frame[org_index+1];
    tmp = tmp + org_frame[org_index+2];

    if(tmp <= this->distance){
        // Es ist ein Hintergrund Pixel
        areas_buffer[area_index]=0;
    } else{
        // Es ist ein Vordergrund-Pixel
        // 'Connected Components Labeling Algorithmus' durchführen
        labeling(area_index);
    }
}
if(i < width-1){// Nächstes Pixel Betrachten
    i = i++;
    org_index = org_index + 3;// + 3 da ein Pixel durch 3 byte dargestellt wird
    area_index = area_index + 1;
} else{
    i = 0; // Bereich, der nicht zum Subframe gehört, überspringen
    org_index = org_index + (WIDTH-width+1)*3;
    area_index = area_index + (WIDTH-width+1);
}
}while(area_index <= x0 + width + WIDTH*(height+y0-1));
// Solange nicht das letzte Subframe Pixel erreicht ist.
// x1 + width bewegt uns auf der x-Achse bis auf den rechten
// Rand des SubFrame. Durch hieght+y1-1 ermitteln wir die Anzahl der Schritte,
// die auf der y-Achse nach unten zu gehen sind und durch die Multiplikation
// Mit WIDTH erhalten wie die letzte Positon (rechts unten) des SubFrame.
// -1 verhindert, dass wir eine Zeile zu tief auskommen.

#ifdef debug==1
    cout << "1. Durchlauf\n";
    for(unsigned int i=0;i<= akt_label;i++){
        cout << i << "\t";
        cout << labels[i][0] << "\t";
        cout << labels[i][1] << "\t";
        cout << labels[i][2] << "\t";
        cout << labels[i][3] << "\t";
        cout << labels[i][4] << "\t";
        cout << labels[i][5] << "\t";
        cout << labels[i][6] << "\t";
        cout << labels[i][7] << "\t";
        cout << labels[i][8] << "\n";
    }
    cout << akt_label << "----\n";
#endif
}
```

Quellcode der Beispielapplikation

```
/**
 * Diese Methode stellt das Thresholding so ein,
 * dass für die drei Farben Rot, Grün, Blau je ein Bereich
 * bestimmt werden kann in dem sich diese Farbe befinden muss.
 * Trifft diese Bedingung für alle drei Farben zu, so
 * wird das gerade betrachtete Pixel als Vordergrundpixel eingestuft.
 */
void o_tracing::set_rgb_threshold(unsigned char red_min, unsigned char red_max,
    unsigned char green_min, unsigned char green_max,
    unsigned char blue_min, unsigned char blue_max){
    this->rgb_tracing=true;
    this->red_min=red_min;
    this->red_max=red_max;
    this->green_min=green_min;
    this->green_max=green_max;;
    this->blue_min=blue_min;
    this->blue_max=blue_max;
}

/**
 * Diese Methode stellt das Thresholding so ein, dass die drei
 * Farbanteile Rot, Grün, Blau eines Pixel zusammenaddiert werden.
 * Ist dieser Wert größer als der als Parameter übergebene
 * Wert, so wird das gerade betrachtete Pixel als Vordergrundpixel eingestuft.
 */
void o_tracing::set_total_threshold(unsigned int distance){
    rgb_tracing=false;
    this->distance=distance;
}

/**
 * Diese Methode startet den Vorgang der Objekterkennung / Objektverfolgung
 * durch das Aufrufen der entsprechenden Methoden.
 */
void o_tracing::start_tracing(unsigned char * in_org_frame){
    org_frame=in_org_frame;
    // Hintergrund ausfiltern und Flächen erkennen.
    threshold();
    join_labels();
    tracing();
    //draw_tracing_frame();
}

/**
 * Diese Funktion errechnet für den angegebenen Index eine
 * Mengendichte.
 * Mengendichte =  $MENGE^2 * 100 / ((X\_MAX - X\_MIN + 1) * (Y\_MAX - Y\_MIN + 1))$ 
 */
unsigned int o_tracing::get_mengendichte(unsigned int index){
    unsigned int tmp1 = (labels[index][MENGE]);
    unsigned int tmp2 = (labels[index][X_MAX]-labels[index][X_MIN]+1)
        * (labels[index][Y_MAX]-labels[index][Y_MIN]+1);
    unsigned int tmp3 = tmp1*100 / tmp2;
    unsigned int tmp4 = tmp3 * tmp1;
    return (tmp4);
}

/**
 * Diese Methode errechnet zusammen mit der Information aus ihrem
 * vorhergehenden Aufruf die Geschwindigkeit und Richtung des zu
 * verfolgenden Objektes.
 * Weiter wird ein Scannfenster errechnet/ausgegeben, in dem das Objekt

```

Quellcode der Beispielapplikation

```
* erwartet wird.
*/
void o_tracing::tracing(){

    static bool first = 1;
    /* static unsigned int old_x_max;
       static unsigned int old_x_min;
       static unsigned int old_y_max;
       static unsigned int old_y_min;
       static unsigned int old_xcenter;
       static unsigned int old_ycenter;*/
    unsigned int xcenter;
    unsigned int ycenter;
    int delta_x;
    int delta_y;
    int tmp_x0, tmp_y0, tmp_x1, tmp_y1;
    unsigned int object_width;
    unsigned int objekt_height;
    // Aussenmaße des Objekts
    object_width=labels[objekt_index][X_MAX]-labels[objekt_index][X_MIN];
    objekt_height=labels[objekt_index][Y_MAX]-labels[objekt_index][Y_MIN];
    if (objekt_index==0){ // Es gibt kein Objekt
        x0=0;
        y0=0;
        width=WIDTH-1;
        height=HEIGHT-1;
    }
    else{
        if ( ( object_width<2) || (objekt_height<2) ){
            x0=0;
            y0=0;
            width=WIDTH-1;
            height=HEIGHT-1;
        }
        else{
            // Mittelung der Objekte
            xcenter = labels[objekt_index][X_MIN] + (object_width/2);
            ycenter = labels[objekt_index][Y_MIN] + (objekt_height/2);
            if (first){
                first = false;
                delta_x = 10;
                delta_y = 10;
            } else {
                // delta auf beiden Achsen
                delta_x = xcenter-old_xcenter;
                delta_y = ycenter-old_ycenter;
            }

            // Errechnung des neuen Scannfenster + Bereichssicherung
            tmp_x0=xcenter - 1 - (object_width);
            if (delta_x<0) tmp_x0 = tmp_x0 + delta_x;
            if (tmp_x0<0) x0=0;
            else x0=tmp_x0;

            tmp_y0=ycenter - 1 - (objekt_height);
            if (delta_y<0) tmp_y0 = tmp_y0 + delta_y;
            if (tmp_y0<0) y0=0;
            else y0=tmp_y0;

            tmp_x1=xcenter + 1 + (object_width);
            if (delta_x>0) tmp_x1 = tmp_x1 + delta_x;
            if (tmp_x1>WIDTH) width=WIDTH-x0-1;
            else width=tmp_x1-x0-1;
        }
    }
}
```

Quellcode der Beispielapplikation

```
        tmp_y1=ycenter + 1 + (objekt_height);
        if (delta_y>0) tmp_y1 = tmp_y1 + delta_y;
        if (tmp_y1>HEIGHT) height=HEIGHT-y0-1;
        else height=tmp_y1-y0-1;

        // Es werden Informationen über das bald alte Objekt abgelegt!
        old_x_max = labels[objekt_index][X_MAX];
        old_x_min = labels[objekt_index][X_MIN];
        old_y_max = labels[objekt_index][Y_MAX];
        old_y_min = labels[objekt_index][Y_MIN];
        old_xcenter= xcenter;
        old_ycenter= ycenter;
    }
}
#ifdef debug
if (height >= HEIGHT)
{
    cout << "!! height hat die Auflösung überschritten - in Methode tracing \n";
}
if (width >= WIDTH)
{
    cout << "!! width hat die Auflösung überschritten - in Methode tracing \n";
}
#endif
}

/**
 * Diese Funktion zeichnet kleine Makierungen in die Ecken des
 * Scannfensters.
 */
void o_tracing::draw_tracing_frame(){

    for(int i=0;i<2;i++)
    {
        unsigned int my_x0=(3*x0)+((id+i)%3);
        if (id>3) cout << "!!" << endl;
        unsigned int my_width=3*WIDTH;

        // Ecke links oben.
        areas_buffer[my_x0+(my_width*y0)]=255;
        areas_buffer[my_x0+(my_width*y0)+3]=255;
        areas_buffer[my_x0+(my_width*y0)+6]=255;
        areas_buffer[my_x0+(my_width*(y0+1))]=255;
        areas_buffer[my_x0+(my_width*(y0+2))]=255;
        //Ecke rechts oben.
        areas_buffer[my_x0+(my_width*y0)+width*3]=255;
        areas_buffer[my_x0+(my_width*y0)+width*3-3]=255;
        areas_buffer[my_x0+(my_width*y0)+width*3-6]=255;
        areas_buffer[my_x0+(my_width*(y0+1))+width*3]=255;
        areas_buffer[my_x0+(my_width*(y0+2))+width*3]=255;
        //Ecke links unten.
        areas_buffer[my_x0+(my_width*(y0+height))]=255;
        areas_buffer[my_x0+(my_width*(y0+height))+3]=255;
        areas_buffer[my_x0+(my_width*(y0+height))+6]=255;
        areas_buffer[my_x0+(my_width*(y0+height-1))]=255;
        areas_buffer[my_x0+(my_width*(y0+height-2))]=255;
        //Ecke recht unten.
        areas_buffer[my_x0+(my_width*(y0+height))+width*3]=255;
        areas_buffer[my_x0+(my_width*(y0+height))+width*3-3]=255;
        areas_buffer[my_x0+(my_width*(y0+height))+width*3-6]=255;
```

Quellcode der Beispielapplikation

```
        areas_buffer[my_x0+(my_width*(y0+height-1))+width*3]=255;
        areas_buffer[my_x0+(my_width*(y0+height-2))+width*3]=255;
    }

}

/**
 * Diese Funktion zeichnet kleine Makierungen in die Ecken des
 * Scannfensters.
 */
void o_tracing::draw_tracing_frame(unsigned char * my_frame){

    for(int i=0;i<2;i++){
        {
            unsigned int my_x0=(3*x0)+((id+i)%3);
            if (id>3) cout << '!' << endl;
            unsigned int my_width=3*WIDTH;

            // Ecke links oben.
            my_frame[my_x0+(my_width*y0)]=255;
            my_frame[my_x0+(my_width*y0)+3]=255;
            my_frame[my_x0+(my_width*y0)+6]=255;
            my_frame[my_x0+(my_width*(y0+1))]=255;
            my_frame[my_x0+(my_width*(y0+2))]=255;
            //Ecke rechts oben.
            my_frame[my_x0+(my_width*y0)+width*3]=255;
            my_frame[my_x0+(my_width*y0)+width*3-3]=255;
            my_frame[my_x0+(my_width*y0)+width*3-6]=255;
            my_frame[my_x0+(my_width*(y0+1))+width*3]=255;
            my_frame[my_x0+(my_width*(y0+2))+width*3]=255;
            //Ecke links unten.
            my_frame[my_x0+(my_width*(y0+height))]=255;
            my_frame[my_x0+(my_width*(y0+height))+3]=255;
            my_frame[my_x0+(my_width*(y0+height))+6]=255;
            my_frame[my_x0+(my_width*(y0+height-1))]=255;
            my_frame[my_x0+(my_width*(y0+height-2))]=255;
            //Ecke recht unten.
            my_frame[my_x0+(my_width*(y0+height))+width*3]=255;
            my_frame[my_x0+(my_width*(y0+height))+width*3-3]=255;
            my_frame[my_x0+(my_width*(y0+height))+width*3-6]=255;
            my_frame[my_x0+(my_width*(y0+height-1))+width*3]=255;
            my_frame[my_x0+(my_width*(y0+height-2))+width*3]=255;
        }
    }

}

/**
 * In diesem Konstruktor werden Vorbereitungen getroffen, die
 * für die Darstellung eines Grafisches Fenster nötig sind.
 */
video_out::video_out(){
    // Vorbereitungen für das X-Fenster
    display_name = getenv("DISPLAY"); // Einholen der Displayadresse
    // Verbindungsaufbau zum X Server
    display = XOpenDisplay(display_name);
    if (display == NULL){
        fprintf(stderr, "Zum Display \"%s\" konnte keiner Verbindung aufgebaut werden.", disp
        exit(1);
    }
    // Ermittle die Abmessungen des (Standard) Desktop
    screen_num = DefaultScreen(display);
    printf("window Breite= %d; Höhe= %d\n", WIDTH, HEIGHT);
    // Erzeuge ein einfaches Fenster das "child" des root Fensters ist.
    // Das Weiß des root wird Hintergrund der child.
```


Quellcode der Beispielapplikation

```
// Das Fenster wird in der linken oberen Ecke plaziert.
win = create_simple_window(display, WIDTH, HEIGHT, 0, 0);
//Events aktivieren
XSelectInput(display, win, ExposureMask | ButtonPressMask |
    KeyPressMask);
// Erzeugen eines graphical context
gc = XCreateGC (display, win, 0, &gcv);
// Windows wird dargestellt auch wenn es noch keinen Inhalt hat.
XMapWindow(display, win);
}

/**
 * Diese Methode überprüft, ob eine Taste betätigt wurde.
 * Wurde eine Taste betätigt, gibt sie true zurück
 * ansonsten false
 */
bool video_out::get_event(unsigned char *buffer, unsigned char *type,
    unsigned char *red, unsigned char *green, unsigned char *blue, KeySym *key){

    if (XCheckMaskEvent(display, ButtonPressMask | KeyPressMask, &an_event)){
        switch (an_event.type) {
            case ButtonPress:
                handle_button_down((XButtonEvent*)&an_event.xbutton, buffer, type, red, green, blue);
                break;
            case KeyPress:
                *key = XLookupKeysym(&an_event.xkey, 0);

                //return(true);
                break;
            default: /* Ignoriere andere Ereignisse */
                break;
        }
        return (true);
    } else{
        return (false);
    }
}

/**
 * Diese Methode gibt die Anzahl von bit zurück, die für die
 * Farbdarstellung eines Bildpunktes verwendet wird.
 */
int video_out::get_depth(){
    return(DefaultDepth(display, screen_num));
}

/**
 * Diese Methode füllt ein Fenster mit dem Inhalt der Integer Werte
 * im Parameter buffer.
 * So kann ein Bild aus dem Speicher auf dem Monitor ausgegeben werden.
 * Bei dem Bild sollte es sich um ein 32 bit Grauwert codiertes Bild handeln.
 * Beachtet werden muss, dass Helligkeitwerte größer 255 (Dezimal)
 * unbestimmt bzw. als Modulo von 255 ausgegeben werden.
 */
void video_out::display_frame (unsigned char * translated_buffer,
    int depth, int bpl, unsigned int *buffer){

    struct imaged{
        unsigned int *buffer;
        unsigned int width;
        unsigned int height;
    };
};
```

Quellcode der Beispielapplikation

```
imagem my_imagem;
my_imagem.width=WIDTH;
my_imagem.height=HEIGHT;
my_imagem.buffer = buffer;
switch(depth) {
/* case 8:{
    int x,y,z,k,pixel;
    for(z=k=y=0;y!=my_imagem.height;y++)
    for(x=0;x!=my_imagem.width;x++)
    {
        // for grayscale-only 8 bit depth
        // can't work in 8 bit color display
        pixel=(my_imagem.buffer[z++]+
        my_imagem.buffer[z++]+
        my_imagem.buffer[z++])/3;
        translated_buffer[k++]=pixel;
    }
}
break;*/
case 8:{
    unsigned x,y,z,k;
    //unsigned buffer;
    for(z=k=y=0;y!=my_imagem.height;y++)
    for(x=0;x!=my_imagem.width;x++){
        if (z == (my_imagem.width*my_imagem.height)) break;
        // alt for 24 bit depth, organization BGRX
        // 24 bit RGBX -> RGB
        translated_buffer[k+0]=my_imagem.buffer[z+0];    // red
        translated_buffer[k+1]=my_imagem.buffer[z+0];    // green
        translated_buffer[k+2]=my_imagem.buffer[z+0];    // blue
        translated_buffer[k+3]=0;
        k+=4; z+=1;
    }
}
break;
case 16:
{
    unsigned int x,y,z,k/*,pixel*/,r,g,b;
    unsigned short *word;
    word=(unsigned short *) translated_buffer;
    for(z=k=y=0;y!=my_imagem.height;y++)
    for(x=0;x!=my_imagem.width;x++){
        if (z == (my_imagem.width*my_imagem.height)) break;
        r=my_imagem.buffer[z++] <<8;
        g=my_imagem.buffer[z++] <<8;
        b=my_imagem.buffer[z++] <<8;
        r &= 0xf800;
        g &= 0xfc00;
        b &= 0xf800;
        word[k++]=r|g>>5|b>>11;
    }
}
break;
case 32:
case 24:{
    unsigned x,y,z,k;
    for(z=k=y=0;y!=my_imagem.height;y++)
    for(x=0;x!=my_imagem.width;x++)
    {
        if (z == (my_imagem.width*my_imagem.height*3)) break;
        // alt for 24 bit depth, organization BGRX
        // 24 bit RGBX -> RGB
```

Quellcode der Beispielapplikation

```

        translated_buffer[k+2]=my_imagem.buffer[z+2];    // red
        translated_buffer[k+1]=my_imagem.buffer[z+1];    // green
        translated_buffer[k+0]=my_imagem.buffer[z+0];    // blue
        k+=4; z+=3;
    }
}
break;
default:
    cout << "'Diese Farbtiefe wird nicht unterstützt !'" << endl;
    break;
}
static int first = 1;
if (first == 1){
    ximage = XCreateImage (display, CopyFromParent, 24 /*zuvor depth*/,
        ZPixmap, 0, (char *)translated_buffer, my_imagem.width,
        my_imagem.height, bpl*8, bpl * my_imagem.width);
    first = 0;
}
//Windows wird dargestellt
XPutImage(display, win, gc, ximage, 0,0,0,0, my_imagem.width, my_imagem.height);
//Windows wird dargestellt
XFlush(display);
//XDestroyImage(ximage);
}

/**
 * Diese Methode füllt ein Fenster mit dem Inhalt der char Werte
 * im Parameter buffer.
 * So kann ein Bild aus dem Speicher auf dem Monitor ausgegeben werden.
 * Bei dem Bild sollte es sich um ein 24 bit RGB Farbcodiertes Bild handeln.
 */
void video_out::display_frame (unsigned char * translated_buffer,
    int depth, int bpl, unsigned char *buffer){
    struct imagem{
        unsigned char *buffer;
        unsigned int width;
        unsigned int height;
    };
    imagem my_imagem;
    my_imagem.width=WIDTH;
    my_imagem.height=HEIGHT;
    my_imagem.buffer = buffer;
    switch(depth) {
        /* case 8:{
            int x,y,z,k,pixel;
            for(z=k=y=0;y!=my_imagem.height;y++)
            for(x=0;x!=my_imagem.width;x++)
            {
                // for grayscale-only 8 bit depth
                // can't work in 8 bit color display
                pixel=(my_imagem.buffer[z++]+
                    my_imagem.buffer[z++]+
                    my_imagem.buffer[z++])/3;
                translated_buffer[k++]=pixel;
            }
        }
        break;*/
        case 8:{
            unsigned x,y,z,k;
            //unsigned buffer;
            for(z=k=y=0;y!=my_imagem.height;y++)
            for(x=0;x!=my_imagem.width;x++){

```

Quellcode der Beispielapplikation

```

        if (z == (my_imagem.width*my_imagem.height)) break;
        // alt for 24 bit depth, organization BGRX
        // 24 bit RGBX -> RGB
        translated_buffer[k+0]=my_imagem.buffer[z+0];    // red
        translated_buffer[k+1]=my_imagem.buffer[z+0];    // green
        translated_buffer[k+2]=my_imagem.buffer[z+0];    // blue
        translated_buffer[k+3]=0;
        k+=4; z+=1;
    }
}
break;
case 16:
{
    unsigned int x,y,z,k/*,pixel*/,r,g,b;
    unsigned short *word;
    word=(unsigned short *) translated_buffer;
    for(z=k=y=0;y!=my_imagem.height;y++)
    for(x=0;x!=my_imagem.width;x++){
        if (z == (my_imagem.width*my_imagem.height)) break;
        // for 16 bit depth, organization 565
        //      fprintf (stdout, '%d - %d\n', (imagem_t.x*imagem_t.y*imagem_t.w), z);
        r=my_imagem.buffer[z++] <<8;
        g=my_imagem.buffer[z++] <<8;
        b=my_imagem.buffer[z++] <<8;
        r &= 0xf800;
        g &= 0xfc00;
        b &= 0xf800;
        word[k++]=r|g>>5|b>>11;
    }
}
break;
case 32:
case 24:{
    //Windows wird dargestellt
    //XMapWindow(display, win);
    unsigned x,y,z,k;
    for(z=k=y=0;y!=my_imagem.height;y++)
    for(x=0;x!=my_imagem.width;x++)
    {
        if (z == (my_imagem.width*my_imagem.height*3)) break;
        // alt for 24 bit depth, organization BGRX
        // 24 bit RGBX -> RGB
        translated_buffer[k+2]=my_imagem.buffer[z+2];    // red
        translated_buffer[k+1]=my_imagem.buffer[z+1];    // green
        translated_buffer[k+0]=my_imagem.buffer[z+0];    // blue
        k+=4; z+=3;
    }
}
break;
default:
    cout << "Diese Farbtiefe wird nicht unterstuetzt !" << endl;
    break;
}
static int first = 1;
if (first == 1){
    ximage = XCreateImage (display, CopyFromParent, 24 /*zuvor depth*/,
        ZPixmap, 0, (char *)translated_buffer, my_imagem.width,
        my_imagem.height, bpl*8, bpl * my_imagem.width);
    first = 0;
}
XPutImage (display, win, gc, ximage, 0,0,0,0, my_imagem.width, my_imagem.height);
XFlush(display);

```

Quellcode der Beispielapplikation

```
//XDestroyImage(ximage);
}

/**
 * Erzeugt ein X-Window-Fenster
 */
Window video_out::create_simple_window(Display* display, int width, int height, int x, int y)
{
    int win_border_width = 2;
    Window win;
    // Erzeuge ein Fenster als Child des root Fenster.
    // Das Schwarz und Weiß des root Fenster werden für den
    // Vorder- und Hintergrund des neuen Fenster verwendet.
    // Das Fenster wird links oben an den Koordinaten 0,0
    // platziert und hat die Ausdehnung WIDTH, HEIGHT
    win = XCreateSimpleWindow(display, RootWindow(display, screen_num),
                              0, 0, WIDTH, HEIGHT, win_border_width,
                              BlackPixel(display, screen_num),
                              WhitePixel(display, screen_num));
    Window root = RootWindow(display, screen_num);
    cout << "Root=" << root << endl;
    // Sende alle Nachrichten und Sorge so dafür, dass das
    // Fenster aktualisiert wird.
    XFlush(display);
    return win;
}

/**
 * Diese Methode behandelt die Mouse-Events.
 * Wird eine Mousetaste im grafischen Fenster gedrückt,
 * werden die Koordinaten und die entsprechenden RGB
 * Farbanteile des Pixels über dem sich die Mouse befindet
 * im Textfenster ausgegeben.
 */
void video_out::handle_button_down(XButtonEvent* button_event, unsigned char *buffer,
    unsigned char *type, unsigned char *red, unsigned char *green, unsigned char *blue)
{
    int x, y; /* invert the pixel under the mouse. */
    //unsigned char red, green, blue;
    x = button_event->x;
    y = button_event->y;
    switch (button_event->button) {
        case Button1:
            *type = 1;
            *blue = buffer[x*3+(y*WIDTH)*3];
            *green = buffer[x*3+(y*WIDTH)*3+1];
            *red = buffer[x*3+(y*WIDTH)*3+2];
            break;
        case Button2:
            *type = 2;
            *blue = buffer[x*3+(y*WIDTH)*3];
            *green = buffer[x*3+(y*WIDTH)*3+1];
            *red = buffer[x*3+(y*WIDTH)*3+2];
            break;
        case Button3:
            *type = 3;
            *blue = buffer[x*3+(y*WIDTH)*3];
            *green = buffer[x*3+(y*WIDTH)*3+1];
            *red = buffer[x*3+(y*WIDTH)*3+2];
            break;
    }
}
```

Quellcode der Beispielapplikation

```
/**
 * Der Konstruktor der Klasse video_in initialisiert einige Variablen
 */
video_in::video_in() {
    frame=0;
}

/**
 * Einen Frame vom Device lesen und in frame 1 oder frame 2 ablegen.
 */
int video_in::grab_frame (/*int frame*/) {
    if (ioctl(video_fd, VIDIOCMCAPTURE, buf + frame) == -1);
    return 0;
}

/**
 * Einen Frame vom Device lesen und in den Parameter frame ablegen.
 */
int video_in::grab_frame (int selected_frame) {
    if (ioctl(video_fd, VIDIOCMCAPTURE, buf + selected_frame) == -1);
    return 0;
}

/**
 * Warten bis der Grabvorgang abgeschlossen ist.
 */
int video_in::grab_sync () {
    if (ioctl (video_fd, VIDIOCSYNC, buf+ (!frame) ) == -1);
    return 0;
    return frame;
}

/**
 * Initialisierung des Grabben von einen Videodevice in eine nmap
 */
int video_in::grab_open (char *device, int width, int height, int depth) {
    unsigned int size=0;
    // Öffnen des Video Devices
    if ((video_fd=open(device, O_RDWR)) < 0 ) return -1;
    // Ermittlung der Videoeinstellungen
    if (ioctl (video_fd, VIDIOCGCAP, &cap) < 0) return -1;
    // Typ der Karte ausgeben
    cout << "Kartentyp: " << cap.name << endl;
    cout << "Maximale Auflösung: " << cap.maxwidth << " x " << cap.maxheight << endl;
    // Test ob die Video-Auflösung vom Gerät erreicht werden kann
    if (width > cap.maxwidth || height > cap.maxheight ||
        width < cap.minwidth || height < cap.minheight)
        return -1;
    // Einstellung der Bildeigenschaften
    // if (ioctl (video_fd, VIDIOCGPICT, &pic) < 0) return -1;
    // chn.type = VIDEO_TYPE_CAMERA;
    // chn.norm = norm;
    // chn.channel = 0;
    // if (ioctl (video_fd, VIDIOCSCHAN, &chn) < 0) return -1;
    // mmap

    // Jeder Buffer bekommt seinen eigenen Speicherbereich zugewiesen.
    // und die Auflösung wird festgelegt.
    buf[0].frame = 0;
    buf[1].frame = 1;
    buf[0].width = buf[1].width = width;
    buf[0].height = buf[1].height = height;
}
```

Quellcode der Beispielapplikation

```
// Abhängig von depth wird das Farbformat eingestellt.
// depth, entspricht der Anzahl der benötigten Bytes.
switch (depth)
{
    case 1:
        buf[0].format = buf[1].format = VIDEO_PALETTE_GREY;
        break;
    case 2:
        buf[0].format = buf[1].format = VIDEO_PALETTE_RGB565;
        break;
    case 3:
    default:
        buf[0].format = buf[1].format = VIDEO_PALETTE_RGB24;
        break;
}

// Ermitteln des Speicherbedarfs für einen Frame.
size = width * height * depth;
// Ermitteln des Speicherbedarfs für alle Frames des Buffers.
mbuf.size = size * VIDEO_BUFFERS;
mbuf.frames = 2;
mbuf.offsets[0] = 0;    // 1. Offset für 1. Frame
mbuf.offsets[1] = size; // 2. Offset für 2. Frame
// Teile dem mmap-Interface die mbuf-Informationen mit.
if (ioctl (video_fd, VIDIOCGMBUF, &mbuf) < 0) return -1;
buffer = (unsigned char *)mmap (0, mbuf.size, PROT_READ | PROT_WRITE, MAP_SHARED, video_fd, 0);
if (buffer == (unsigned char *) -1) return -1;
return 0;
}

/**
 * Diese Methode sorgt dafür, dass ein Frame eingefangen
 * und der Frame im anderen Buffer synchronisiert wird.
 */
unsigned char * video_in::grab_pix (void) {

    if (grab_frame() < 0) return 0;
    if (grab_sync() < 0) return 0;
    frame=!frame;
    return buffer + mbuf.offsets[frame]; // gibt einen Frame zurück
}

/**
 * Der Speicher, der für das Grabben verwendet wurde, wird wieder freigegeben
 * und der File Descriptor des Video Device wird wieder freigegeben.
 */
void video_in::grab_close (void) {
    munmap (buffer, mbuf.size);
    close(video_fd);
}

/**
 * Diese Methode stellt mit die rot grün und blau min. max. Werte mit hilfe
 * der Toleranz ein.
 */
void settoleranz(unsigned char red, unsigned char green, unsigned char blue,
    unsigned char *red_min, unsigned char *red_max, unsigned char *green_min,
    unsigned char *green_max, unsigned char *blue_min, unsigned char *blue_max, unsigned char *toleranz) {
    if (red <= 255 - toleranz) *red_max = red + toleranz;
    else *red_max = 255;
    if (red >= toleranz) *red_min = red - toleranz;
```

Quellcode der Beispielapplikation

```
        else *red_min = 0;
        if (green<=255-toleranz) *green_max = green + toleranz;
        else *green_max = 255;
        if (green>=toleranz) *green_min = green - toleranz;
        else *green_min = 0;
        if (blue<=255-toleranz) *blue_max = blue + toleranz;
        else *blue_max = 255;
        if (blue>=toleranz) *blue_min = blue - toleranz;
        else *blue_min = 0;
    }

/**
 * Diese Methode erzeugt auf er Console ein Menu und übernimmt das Auswerten
 * von Tastatur und Mouseeingaben von Grafischen Fenster.
 */
bool Menue(video_out *out, o_tracing *Objekt1, o_tracing *Objekt2, o_tracing *Objekt3, unsigned char *red, unsigned char *green, unsigned char *blue, bool &grab1, bool &grab2, bool &grab3, bool &timeing){
    static bool first=true;
    static unsigned char toleranz1 = 35;
    static unsigned char toleranz2 = 35;
    static unsigned char toleranz3 = 35;
    unsigned char type, red, green, blue, red_min, red_max;
    unsigned char green_min, green_max, blue_min, blue_max;
    KeySym key;
    bool break_loop = false;
    type = 0;
    if ( out->get_event(org_frame, &type, &red, &green, &blue, &key) || first ){
        if (type==0){// Tastatur
            //cout << key << endl;
            switch (key){
                case 'b':
                case 'B':
                    break_loop = true;
                    break;
                case '1':
                    grab1 = ! grab1;
                    break;
                case '2':
                    grab2 = ! grab2;
                    break;
                case '3':
                    grab3 = ! grab3;
                    break;
                case 't':
                case 'T':
                    timeing = ! timeing;
                    break;
                case 'q':
                case 'Q':
                    if (toleranz1<(255/2)) toleranz1++;
                    break;
                case 'a':
                case 'A':
                    if (toleranz1>0) toleranz1--;
                    break;
                case 'w':
                case 'W':
                    if (toleranz2<(255/2)) toleranz2++;
                    break;
                case 's':
                case 'S':
                    if (toleranz2>0) toleranz2--;
```


Quellcode der Beispielapplikation

```

        break;
        case 'e':
        case 'E':
            if (toleranz3<(255/2)) toleranz3++;
            break;
        case 'd':
        case 'D':
            if (toleranz3>0) toleranz3--;
            break;

        default:
            break;
    }
} else { // Mousetaste
    // min. max. Werte bestimmen
    switch (type){
        case 1:
            settoleranz(red, green, blue, &red_min, &red_max, &green_min, &green_max, &blue_min, &blue_max);
            Objekt1->set_rgb_threshold(red_min, red_max, green_min, green_max, blue_min, blue_max);
            break;
        case 2:
            settoleranz(red, green, blue, &red_min, &red_max, &green_min, &green_max, &blue_min, &blue_max);
            Objekt2->set_rgb_threshold(red_min, red_max, green_min, green_max, blue_min, blue_max);
            break;
        case 3:
            settoleranz(red, green, blue, &red_min, &red_max, &green_min, &green_max, &blue_min, &blue_max);
            Objekt3->set_rgb_threshold(red_min, red_max, green_min, green_max, blue_min, blue_max);
            break;
        default:
            break;
    }
} // end else
//clear();
cout << endl << "Diplomarbeit - \"Kameragestützte Echtzeit Objekterkennung unter Linux\"";
cout << "von Thomas Maurer Version 1.1" << endl << endl;
cout << "Für Eingaben muss der Fokus auf dem Grafischen-Fenster liegen!" << endl;
cout << "1 = Captureframe 1 ein/aus ";
if (grab1) cout << "[EIN] ";
else cout << "[AUS] ";
cout << "Toleranz 1 = [" << (toleranz1*2) << "] q=+2 / a=-2 und Mousebutton1" << endl;
cout << "2 = Captureframe 1 ein/aus ";
if (grab2) cout << "[EIN] ";
else cout << "[AUS] ";
cout << "Toleranz 2 = [" << (toleranz2*2) << "] w=+2 / s=-2 und Mousebutton2" << endl;
cout << "3 = Captureframe 1 ein/aus ";
if (grab3) cout << "[EIN] ";
else cout << "[AUS] ";
cout << "Toleranz 3 = [" << (toleranz3*2) << "] e=+2 / d=-2 und Mousebutton3" << endl;
cout << "t = Timing ein/aus ";
if (timeing) cout << "[EIN] " << endl;
else cout << "[AUS] " << endl;
cout << "b = Programm beenden." << endl << endl;
cout << "Mit Mausknopf 1-3 kann je für Frame 1-3 die Threshold-Farbe gewählt werden." << endl;
if (!first && (type==1 || type==2 || type==3)) {
    cout << "Button ";
    printf("%i: RGB=[%i][%i][%i]\n", type, red, green, blue);
}
#if debug==3
    printf("%iR%i %iG%i %iB%i\n", red_min, red_max, green_min, green_max, blue_min, blue_max);
#endif
first = false;
} //end if

```

Quellcode der Beispielapplikation

```
        return (break_loop);
    }

/**
 * Programmeinstieg
 */
int main (int argc, char* argv[]) {

    bool break_loop = false;
    unsigned char * translated_buffer;
    unsigned char * org_frame;// Zeiger auf den eingefangenen Frame

    bool grab1 = true;
    bool grab2 = true;//false;
    bool grab3 = true;//false;
    bool timing = true; // wenn false muss die Destruktion
        // angepasst werden -> sonst Segmentation fault
    bool retime = false; // Wird das Timing neu gestartet wird der
        // erste verfälschte Wert verworfen.
    int depth;// Farbtiefe des Bildes
    int bpl;// Anzahl von Bytes, die für die Farbtiefe benötigt werden
    timeval *tv1, *tv2;// Variablen zur Zeiterfassung
    int a;// Laufvariable
    char device[200];//="/dev/video";

    switch (argc){
        case 1:
            strcpy(device, "/dev/video");
            cout << "Keine Parameterangabe es wird \"/dev/video\" verwendet." << endl;
            break;
        case 2:
            strcpy(device, argv[1]);
            break;
        default:
            cout << "Als Parameter ist nur das Video device erlaubt z.B: \"/dev/video1\""" << endl;
            exit(0);
            break;
    }

    video_out * out = new video_out();
    depth = out->get_depth();
    // Farbtiefe ermitteln
    cout << "Farbtiefe= " << depth << " Bit" << endl;
    // Abhängig von der Farbtiefe müssen später 1 bis 4 Byte pro Bildpunkt
    // allokiert werden.
    switch(depth){
        case 24:
            bpl=4;
            break;
        case 16:
        case 15:
            bpl=2;
            break;
        default:
            bpl=1; break;
    }
    // Speicher für zu wandelndes Bild reservieren
    translated_buffer=(unsigned char *)malloc(bpl*WIDTH*HEIGHT);
    // Vorbereitung zum Einfangen von Frames
    video_in * in = new video_in();
    if (in->grab_open(device, WIDTH, HEIGHT, DEPTH) < 0){
        fprintf (stderr, "Device \"%s\" kann nicht geöffnet werden!\n", device);
    }
}
```

Quellcode der Beispielapplikation

```
    exit(0);
}
// Grabbe schon mal einen ungeraden Frame
if (in->grab_frame(1) < 0) return 0;

o_tracing *Objekt1 = new o_tracing();
o_tracing *Objekt2 = new o_tracing();
o_tracing *Objekt3 = new o_tracing();
// Einstellung des Thresholding Verfahren.
    Objekt1->set_total_threshold(255*3-100);
    Objekt2->set_total_threshold(255*3-100);
    Objekt3->set_total_threshold(255*3-100);
//Objekt1->set_rgb_threshold(150, 255, 0, 25, 0, 25);

// Schleife in der ein Bild gegrabbt und danach abgelegt wird.
while (!break_loop){
    if (timeing) gettimeofday(tv1 = new timeval, new timezone);
    for (a=0; a< 100 && break_loop==false; a++)
    {
        // Abwechselnd in die beiden Frame Buffer grabben.
        // und je den anderen eingefangenen Frame freigeben.
        org_frame = in->grab_pix();
        if (grab1){
            Objekt1->start_tracing(org_frame);
            Objekt1->draw_tracing_frame(org_frame);
        }
        if (grab2){
            Objekt2->start_tracing(org_frame);
            Objekt2->draw_tracing_frame(org_frame);
        }
        if (grab3){
            Objekt3->start_tracing(org_frame);
            Objekt3->draw_tracing_frame(org_frame);
        }

        break_loop = Menue(out, Objekt1, Objekt2, Objekt3, org_frame, grab1, grab2, grab3, ti

        // den Frame in X-Windows darstellen.
        out->display_frame(translated_buffer, depth, bpl, org_frame);
        //out->display_frame(translated_buffer, 8, bpl, Objekt1->areas_buffer);

        //for (unsigned int i=0;i<WIDTH*HEIGHT;i++){
        //    Objekt1->areas_buffer[i]=0;
        //}
    }
    gettimeofday(tv2 = new timeval, new timezone);
    if (timeing) {
        //cout << "sec: " << tv2->tv_sec - tv1->tv_sec << " ";
        //cout << "usec: " << tv2->tv_usec - tv1->tv_usec << endl;
        if (!retime) cout << "Sekunden für 100 Bilder: " << (float)(tv2->tv_sec - tv1->tv_s
        retime = false;
    }
    else{
        retime = true;
    }
}
//Aufräumarbeit
in->grab_close();
// Reservierten Speicher wieder freigeben.
delete tv1;
delete tv2;
delete Objekt1;
```

Quellcode der Beispielapplikation

```
delete in;  
delete out;  
free((void *)translated_buffer);  
}
```

[Zurück](#)

Bedienung der Beispielapplikation

[Zum Anfang](#)

[Nach vorne](#)

Rechtliches / Eingetragene
Warenzeichen

Anhang D. Rechtliches / Eingetragene Warenzeichen

Das Landgericht Hamburg hat entschieden, daß durch die Anbringung von Links die Inhalte der gelinkten Seite ggf. mit zu verantworten sind. Dies kann verhindert werden, indem sich der Betreiber der Seite, auf der die Links angebracht sind, ausdrücklich von diesen Inhalten distanziert. Da eine Publikation dieser Arbeit im Internet vorgesehen ist, erkläre ich hiermit:

Ich distanzieren mich vorsorglich von allen Inhalten aller verlinkten Seiten und mache mir deren Inhalte nicht zu Eigen. Ich bin selbstverständlich für jeden Hinweis dankbar, der auf einen Link aufmerksam macht, dessen Inhalt gegen geltendes Recht verstößt, rechtsextremistisch, rassistisch oder sonstige Gesetze und/oder Vorschriften verletzt. Ich werde nach einem entsprechenden Hinweis diesen Link aus dieser Arbeit entfernen.

- "c't"

ist ein eingetragenes Warenzeichen von Verlag Heinz Heise GmbH & Co KG.

- "Linux"

ist ein eingetragenes Warenzeichen von Linus Torvald.

- "Macromedia Flash"

ist ein eingetragenes Warenzeichen der Firma Macromedia Inc.

- "3D PROPHET II MX DUAL-DISPLAY VIDEO"

ist ein eingetragenes Warenzeichen der Firma Hercules c/o Guillemot GmbH.

- "NVIDIA", "GeForce", "GeForce2" und "GeForce2 MX"

ist ein eingetragenes Warenzeichen von der Firma NVIDIA Corporation.

- "Sony EVI D30/31"

ist ein eingetragenes Warenzeichen der Firma Sony Deutschland GmbH.

- "ToUCam Pro"

ist eingetragenes Warenzeichen der Firma Philips GmbH.

- "TerraCAM USB Pro"

ist eingetragenes Warenzeichen der Firma TerraTec Electronic GmbH.

- "WinTV"

ist eingetragenes Warenzeichen der Firma Hauppauge Computer Works, Inc.

- "Windows ME" und "Windows XP"

sind eingetragene Warenzeichen der Firma Microsoft GmbH.

[Zurück](#)

Quellcode der Beispielapplikation

[Zum Anfang](#)

[Nach vorne](#)

Glossar

Glossar

A

API

Steht für engl. Application Programming Interface. Bezeichnung einer Softwareschnittstelle, die für eine problemlose Zusammenarbeit der einzelnen Module innerhalb eines Betriebssystems sorgt. Wichtig ist das diese Schnittstelle einheitlich gehalten wird. Sollte sich die Implementierung hinter der Schnittstelle verändern ist es wünschenswert das dies keinen Einfluss auf die Schnittstelle hat.

C

CCD

CCD steht für engl. charge-coupled device, also ein ladungsgekoppeltes Bauelement. Bei diesen Halbleiterbauelementen werden Informationen in Form von elektrischen Ladungen gespeichert, die später ausgelesen werden können. Durch die Lichtempfindlichkeit der Bauelemente eignen sie sich besonders für Bildsensoren. Gegenüber C-MOS Sensoren lassen sich mit CCD Sensoren bessere Bildraten (Bilder pro Sekunde) erreichen. Weiter treten bei der CCD Technik geringere Schliereneffekte auf. Dafür ist der Preis eines solchen Sensors erheblich höher als der einer vergleichbaren C-MOS Ausführung.

Siehe auch: C-MOS.

C-MOS

C-MOS steht für engl. complementary metal oxide semiconductor. Es handelt sich hierbei um einen Halbleiter, der nach einer verbreiteten und günstigen Produktionstechnik erzeugt wurde. Als Video-Sensor eingesetzt, handelt es sich um eine lichtempfindliche Diode bei der der Spannungsabfall am Bauelement von der einfallenden Lichtmenge abhängig ist. Diese Bauelemente sind deutlich billiger als die auf CCD Technologie basierenden Sensoren. Sie liefern aber im Augenblick noch schlechtere Bildraten (Bilde pro Sekunde) und neigen zu einem stärkeren Schliereneffekt.

Siehe auch: CCD.

D

DMA

DMA steht für engl. direct memory access. Dieser direkte Speicherzugriff sorgt für einen Datenaustausch zwischen Speicher und Peripherie eines Computers unter Umgehung der CPU. Bei diesem Vorgang gibt die CPU kurzzeitig die Kontrolle über den BUS und das an dem BUS angeschlossene Gerät ab, welches dann den Datentransfer durchführt.

F

FBAS-Signal

FBAS-Signal (oder Composit Signal) steht für Farb-Bild-Austast-Synchron-Signal. Es handelt sich hierbei um ein farbiges Videosignal mit Austast- und Synchronimpulsen, das aus der PAL-Codierung der Farbwertsignale entsteht. Beim Fernsehbild muss jeder einzelne Bildpunkt durch Spannungswerte beschrieben werden. Hierfür wurde das FBAS-Signal entwickelt. Die Buchstaben FBAS stehen dabei für bestimmte Merkmale des Signals.

- ◊ F - Farbsignal: Das Farbsignal enthält die Farben Rot, Grün, Blau (Das RGB-Farbsystem).
- ◊ B - Bildsignal: Die Helligkeit eines jeden Punktes wird durch eine Spannung festgelegt.
- ◊ A - Austastsignal: Um den Zeilenrücklauf und Vertikalrücklauf zu definieren.
- ◊ S - Synchronisation: Synchronisierzeichen zwischen Sender und Empfänger.

Siehe auch: PAL.

Frame-Grabber-Karte

Eine Frame-Grabber-Karte hat die Aufgabe das Bild, das von einer geeigneten Videoquelle kommt, in ein für den PC brauchbares Format zu wandeln und zusammen mit einem Gerätetreiber die Bildinformation in den Hauptspeicher des Rechners zu übertragen. Der HF-Tuner über den fast alle Karten verfügen, wandelt das analoge FBAS-Signal in ein YUV-Signal. Dieses Signal wird dann über den BUS in den Speicher geschrieben. Bei der Wahl der Ausgabetechnik stehen zwei Verfahren zur Verfügung. Ist die Auflösung kleiner als der PAL Standard, kann das Bild von der Frame-Grabber-Karte skaliert werden und über die Grafikkarte auf dem Monitor ausgegeben werden. Diese Technik hat den Vorteil, dass sie von den meisten Grafikkarten unterstützt wird. Die Alternative ist, dass das Bild von der Grafikkarte skaliert und auf dem Monitor ausgegeben wird. Dieses Overlay-Verfahren erlaubt beliebige Auflösungen. Der Nachteil ist, dass die Grafikkarte belastet wird und nicht alle Grafikkarten-Modelle diese Technik unterstützen.

Die am weitesten verbreitete Video Capture Chip's BT848, BT878 oder BT879 werden von der Firma Connexant (früher Brooktree) vertrieben. Einige Karten sind zusätzlich mit einem Tuner für Radioempfang ausgestattet.

Siehe auch: PAL, FBAS.

G

GUI

GUI steht für engl. Graphical User Interface. Diese grafische Benutzerschnittstelle ermöglicht dem Benutzer mit dem Programm zu interagieren oder bietet dem Programm die Möglichkeit, Information an den Benutzer auszugeben. Ein typisches Merkmal des GUI sind Fenster, Buttons und Bildschirm-Menüs, die mit der Maus bedient werden können.

I

ISS

ISS steht für engl. International Space Station. Bei diesem weltweiten Partnerprojekt wird die größte und komplizierteste Orbitalstation der Geschichte, in der Umlaufbahn der Erde zusammengebaut. Ist die ab 2002 nutzbare Station im Jahre 2005 fertig gestellt, hat sie voraussichtlich die Größe eines

Fussballfeldes incl. Randbereiche und wiegt ca. 460 Tonnen. Am Bau und der Nutzung der Station sind die USA, Europa, Kanada, Rußland und Japan beteiligt.

N

NTSC

NTSC steht für National Television Standards Committee und wird im asiatischen Raum und Amerika verwendet. Die Auflösung beträgt maximal 640x480 Punkte, wobei die vertikale Auflösung "interlaced" ist. Das heißt, die vertikalen Pixel werden in zwei Durchgängen dargestellt. Zuerst werden alle geraden, danach alle ungeraden vertikalen Zeilen gesendet. Dadurch ergeben sich 59.94 Halbbilder pro Sekunde (59.94 Hz) oder 29.97 Vollbilder.

Siehe auch: PAL, SECAM.

P

PAL

Die Buchstaben PAL stehen für engl. phase alternation line. Dieses in Westeuropa (ausser Frankreich, dort wird SECAM verwendet) und Australien eingesetzte Fernsehnorm wurde von W. Bruch/Telefunken entwickelt und ergänzt das US-amerikanische NTSC-System. Das System ist unempfindlicher gegen Phasenfehler (zeilenweise Umpolung) und verfügt über Farbtinstabilisierung bei örtlichen Empfangsstörungen. PAL hat eine Auflösung von bis zu 768x576 Punkten, wobei die vertikale Auflösung "interlaced" ist. Das heißt, die vertikalen Pixel werden in zwei Durchgängen dargestellt. Zuerst werden alle geraden, danach alle ungeraden vertikalen Zeilen gesendet. Dadurch ergeben sich 50 Halbbilder pro Sekunde (50 Hz) oder 25 Vollbilder.

Siehe auch: SECAM, NTSC.

R

Rauschen

Das Rauschen in der Elektrotechnik wird durch die unregelmäßigen thermischen Bewegungen der Elektronen in Leitern und Halbleitern hervorgerufen. Dadurch ändert sich bei einer Videoaufnahme ständig die Werte für ein einzelnes Pixel in geringen Grenzen. Stärker tritt diese Störung bei unzureichender Ausleuchtung des Objekts auf, da der gleichbleibende Anteil dieser Störung sich mit einem nun schwachen Bildsignal überlagert. Wird dieses Signal nun wieder zu einem helleren Bild verstärkt, enthält es einen höheren Rauschanteil als ein Objekt mit optimaler Ausleuchtung. Durch die Wahl hochwertiger Videokameras lässt sich dem Rauschen in gewissen Grenzen entgegenwirken.

S

SECAM

SECAM ist die Abkürzung für fr. système en couleur avec mémoire, ein Fernseh/Videostandard, der in Frankreich und osteuropäischen Ländern gebräuchlich ist. Von der Technik und Auflösung ist es ähnlich zum PAL-System, mit dem Unterschied, dass die Farbdifferenzsignale zeilenweise nacheinander und nicht gleichzeitig übertragen werden.

Siehe auch: PAL, NTSC.

S-Video

S-Video ist die Abkürzung für Super-Video oft auch als Y/C Video bezeichnet, wobei Y für Luminanz (Helligkeit) und C für Chrominanz (Farbton und Sättigung) steht. Bei diesem Verfahren der Videosignalübertragung wird die Videoinformation in zwei Signale unterteilt: Zum Einen das Farbsignal (chrominance) und zum anderen die Helligkeit (luminance). Durch die Übertragung der beiden getrennten Signale an z.B. einen Fernseher lässt sich eine bessere Bildqualität erzielen als bei der Verwendung eines Composite Video Signals, bei dem die Videoinformationen über nur eine Leitung übertragen werden und in Fernseher wieder zu den Chrominance und Luminanz Bestandteilen aufbereitet werden müssen.

System-Call

Unter einem System-Call versteht man einen Systemaufruf, mit dem ein Prozess spezielle Betriebssystemleistung anfordern kann. Ein Systemcall führt dazu, dass die Registerinhalte des laufenden Prozesses gesichert werden und der laufende Prozess unterbrochen wird, bevor die Befehle des Systemcalls abgearbeitet werden. Parametrisiert wird der Systemcall durch Register und Daten auf dem Stack. Nach erfolgter Abarbeitung wird der Zustand der Register wieder hergestellt und der normale Programmablauf fortgesetzt.

U

USB

USB steht für engl. Universal Serial Bus. Es handelt sich hierbei um eine Anschlussnorm, die den Anschluss externer PC Komponenten wie Videokameras, Scanner, Tastaturen, Audio-Lautsprecher etc. erleichtern soll. Die Daten werden seriell, an bis 127 Geräte, über ein verdrehtes Zweidrahtkabel übertragen. Neben diesen Signalen existieren zwei weitere Leitungen, die 5 Volt Betriebsspannung und Masse führen, so dass beispielsweise Scanner oder Kameras ohne zusätzliche Stromversorgung betrieben werden können. Bei dem USB handelt es sich um viele Punkt-zu-Punkt-Verbindungen, die in mehreren Ebenen sternförmig angeordnet sind. Dadurch ist der Begriff BUS hier nicht ganz zutreffend. Weiter gibt es Mechanismen, die es erlauben, im laufenden Rechner-Betrieb Geräte zu entfernen und hinzuzufügen (Hotplugging). Die zwei zur Zeit existierenden Versionen unterscheiden sich in der Übertragungsrate. Mit 12 MBit/s ist Version 1.1 die zur Zeit gebräuchliche. Version 2.0 verfügt mit 480 MByte/s über deutlich mehr Performance, die gerade in der Bildübertragung benötigt wird. Der USB wurde 1993 von Intel und Microsoft sowie Compaq, DEC, IBM PC Company, NEC und Northern Telecom definiert.

Logisches UND und ODER

Unter einer logischen UND-Verknüpfung versteht man, dass alle Bedingungen, die durch das UND verbunden sind erfüllt sein müssen, um das entsprechende Ausgangsergebnis zu erlangen. Bei einer logischen ODER-Verknüpfung, reicht es, wenn mindestens eine der durch das ODER verbundene Bedingungen erfüllt wird, um das Ausgangsergebnis zu erzielen. Im Gegensatz zum Sprachlichen oder darf auch mehr als nur eine Bedingung erfüllt sein.

P

Video4Linux

Bei Video For Linux, Video4Linux oder kurz v4l handelt es sich um eine API mit der Applikationen auf, an den PC angeschlossene Video Devices (TV-Karten, Kameras, Videotextkarten etc.), zugreifen können. Diese API wird durch das Aufrufen ihrer System-Calls angesprochen. Sie liegt zu Zeit in zwei Versionen vor. Version 1.0, die in dieser Arbeit verwendet wird und eine Version 2.0, die sich in der Entwicklung befindet.

Glossar

Siehe auch: API, System-Call.

Zurück

Rechtliches / Eingetragene

Warenzeichen

Zum Anfang

Nach vorne

Literatur

Literatur

Cox2000 *Video4Linux Programming* Alan Cox Zugriff: März. 2002
<http://kernelnewbies.org/documents/kdoc/videobook/v4lguide.html>

FiPeWaWo2000 *Hypermedia Image Processing Reference* Robert Fisher, Simon Perkins, Ashley Walker und Erik Wolfart Zugriff: Feb. 2002 <http://www.dai.ed.ac.uk/HIPR2/index.html>

Hartl2002 *Licht und Farbe, Teil 6: CIE-XYZ-Farbraum* Stephan Hartl CIE-Farbräume (Undatiert) Zugriff: März 2002 <http://www.copyshop-tips.de/luf06.htm>

YoGeV11999 *Image Processing Fundamentals* I.T. Young, J.J. Gerbrands und L.J. van Vliet Zugriff: Feb. 2002 <http://www.ph.tn.tudelft.nl/Courses/FIP/index.html>

Möbes1999 *Computer Graphics WS 98/99* Prof. Dr. Jan Möbes Vorlesungsskript zur Veranstaltung "Grafische Bildverarbeitung" an der Hochschule Niederrhein September 1999
http://vireo.gatech.edu/ebt-bin/nph-dweb/dynaweb/SGI_Developer/XLib_PG/

Nye1992 *Volume 1: Xlib Programming Manual* Adrian Nye O'Reilly & Associates, Inc. 3. Auflage July 1992
Zugriff: März 2002 http://vireo.gatech.edu/ebt-bin/nph-dweb/dynaweb/SGI_Developer/XLib_PG/

Peeck2002 *Fern-Seher - 21 Webcams mit und ohne Fotofunktion* Dr. Klaus Peeck Verlag Heinz Heise GmbH & Co. KG - Magazin c't 8/2002 S.146-163 April 2002

Quade2001 *Material zur Vorlesung Echtzeitsysteme 1 im Studienfach Technische Informatik an der Hochschule Niederrhein*. Prof. Dr. Jürgen Quade Zugriff: Feb. 2002
<http://ezs.kr.hs-niederrhein.de/lectures/ezs/html/book1.html>

RiCa1978 *Picture thresholding using an iterative selection method*. T.W. Ridler und S. Calvard IEEE Transactions on Systems, Man and Cybernetics, SMC-8:630-632 August 1978

ScoBew1997 *Spectral Selectivity* Ed Scott und Hollis Bewley photo.net 2002 Zugriff: März 2002
<http://www.photo.net/photo/edscott/spectsel.htm>

Literatur

Seilnacht2002 *Lexikon der Farbstoffe und Pigmente - Kapitel Farben* Thomas Seilnacht
www.seilnacht.tuttlingen.com (Undatiert) Zugriff: März 2002
<http://www.seilnacht.tuttlingen.com/Lexikon/Farbe.htm>

UniWup1998 *Auge - Das visuelle System* Autor unbekannt Skriptsammlung der Uni Wuppertal 1998 Zugriff:
März 2002 http://www.stud.uni-wuppertal.de/~ya0023/phys_psy/auge.htm

Zurück
Glossar

Zum Anfang

Nach vorne
Stichwortverzeichnis

Stichwortverzeichnis

4-connectivity, [Findung der Nachbarn](#), [V-Problematik](#), [Algorithmen](#)
8-connectivity, [Übersicht der Implementierten Techniken](#), [Findung der Nachbarn](#), [V-Problematik](#), [Algorithmen](#)
[Übersicht der Implementierten Techniken](#), [Übersicht der Implementierten Techniken](#)
Ableitung
 erste, [Andere Arten der Segmentierung](#)
 zweite, [Andere Arten der Segmentierung](#)
Abszisse, [Verarbeitungsrichtung](#), [Schwellwertbestimmung mit Hilfe von Histogrammen](#), [Automatische Schwellwertbestimmung](#)
Adaptives Thresholding, [Adaptives Thresholding](#), [Weiterentwicklung und Verbesserung](#)
Additive Farbmischung, [Das RGB-Farbsystem](#), [Das CIE-Farbsystem](#)
Algorithmus, [Einleitung](#), [Automatische Schwellwertbestimmung](#), [Findung der Nachbarn](#), [V-Problematik](#), [Kreisflächenähnlichkeit](#), [Algorithmen](#), [Tests](#), [Weiterentwicklung und Verbesserung](#)
Anfrage, [X-Window-Nachrichten](#)
Antwort, [X-Window-Nachrichten](#)
API, [Aufbau einer Objekterkennung](#), [Grabbing](#), [Glossar](#)
Aufbau einer Objekterkennung, [Aufbau einer Objekterkennung](#)
Auge, [Vorwort](#), [Farben](#), [Das Auge verglichen mit einer Kamera](#), [Das Erkennen von Farben im Auge](#), [Das CIE-Farbsystem](#)
Automatische Schwellwertbestimmung, [Automatische Schwellwertbestimmung](#)
Beispielapplikation, [Einleitung](#), [Die Beispielapplikation](#), [Der Datenfluss](#), [Übersicht der Implementierten Techniken](#), [Grabbing](#), [Statisches Thresholding](#), [Flächenerkennung](#), [Randbehandlung](#), [V-Problematik](#), [Objektbewertung](#), [Mengendichte](#), [Kreisflächenähnlichkeit](#), [Subframing / Verfolgung](#), [Grafisches-Fenster](#), [X-Window-Nachrichten](#), [Xlib-Fenster](#), [System-Auslastung](#), [Methoden-Auslastung](#), [Algorithmen](#), [Tests](#), [Weiterentwicklung und Verbesserung](#), [Zusammenfassung und Ausblick](#), [Konfiguration des Systems](#), [Bedienung der Beispielapplikation](#), [Quellcode der Beispielapplikation](#)
Beleuchtung, [Beleuchtung](#)
Blende, [Das Auge verglichen mit einer Kamera](#)
C-MOS, [Das Auge verglichen mit einer Kamera](#), [Aufbau einer Objekterkennung](#), [System-Auslastung](#), [Glossar](#)
CCD, [Das Auge verglichen mit einer Kamera](#), [Aufbau einer Objekterkennung](#), [Glossar](#)
Chow, [Adaptives Thresholding](#)
Chrominanz, [Glossar](#)
CIE
 Farbsystem, [Das RGB-Farbsystem](#), [Das CIE-Farbsystem](#)
 Normfarbtafel, [Das CIE-Farbsystem](#)
 System, [Das CIE-Farbsystem](#)
Ciliarmuskel, [Das Auge verglichen mit einer Kamera](#)
Client, [X-Window-Client-Server-Prinzip](#), [X-Window-Nachrichten](#), [Xlib-Fenster](#)
cones, [Das Erkennen von Farben im Auge](#)
CONNECTIONS, [Der Datenfluss](#), [Übersicht der Implementierten Techniken](#), [Kreisflächenähnlichkeit](#), [Die Selektion](#), [Tests](#)
Convolution, [Andere Arten der Segmentierung](#)
Cornea, [Das Auge verglichen mit einer Kamera](#)
CPU, [2. Echtzeitbedingung - Auslastung](#), [Aufbau einer Objekterkennung](#), [Grabbing](#), [System-Auslastung](#), [Glossar](#)

Stichwortverzeichnis

Datenfluss, Der Datenfluss
Datenflussdiagramm, Der Datenfluss
Dichte, Mengendichte, Kreisflächenähnlichkeit
Differenzbildung, Das Erkennen von Farben im Auge, Differenzbildung, Zusammenfassung und Ausblick
Diskussion und Bewertung, Diskussion und Bewertung
Display Frame, Der Datenfluss
DMA, Aufbau einer Objekterkennung, Glossar
Echtzeit, Echtzeit, Segmentierung
 Objekterkennung, Andere Arten der Segmentierung
 Objektverfolgung, Einleitung, Echtzeit, Zusammenfassung und Ausblick
 Rechner, 2. Echtzeitbedingung - Auslastung
 System, Einleitung, 2. Echtzeitbedingung - Auslastung
Echtzeitbedingung, Einleitung
 erste, 1. Echtzeitbedingung - Pünktlichkeit, Subframing / Verfolgung
 hart, Harte und weiche Echtzeitbedingung
 weich, Harte und weiche Echtzeitbedingung
 weiche, Grabbing
 zweite, 2. Echtzeitbedingung - Auslastung, Subframing / Verfolgung
Echtzeitbedingungen, Echtzeit
Einleitung, Einleitung
Ereignis, 1. Echtzeitbedingung - Pünktlichkeit, X-Window-Nachrichten, Xlib-Fenster
 Behandlung, Xlib-Fenster
error, X-Window-Nachrichten
event, X-Window-Nachrichten
Farben, Farben, Das Erkennen von Farben im Auge, Das RGB-Farbsystem, Das CIE-Farbsystem, Übersicht der Implementierten Techniken, Statisches Thresholding, Schwellwertbestimmung mit Hilfe von Histogrammen, Andere Arten der Segmentierung, Objektbewertung, Die Selektion, Algorithmen, Zusammenfassung und Ausblick, Glossar
 virtuelle, Das CIE-Farbsystem
Fehler, Kreisflächenähnlichkeit, Die Selektion, X-Window-Nachrichten, Algorithmen
Feldgröße, Adaptives Thresholding
File Deskriptor, Grabbing
Filter, Farben, Statisches Thresholding, Schwellwertbestimmung mit Hilfe von Histogrammen, Andere Arten der Segmentierung, Differenzbildung, Konfiguration des Systems
Findung der Nachbarn, Findung der Nachbarn
Flächenerkennung, Der Datenfluss, Übersicht der Implementierten Techniken, Verarbeitungsrichtung, Flächenerkennung, V-Problematik, Objektbewertung, Mengendichte
Fovea, Das Erkennen von Farben im Auge
Frame-Grabber-Karte, Aufbau einer Objekterkennung, Grabbing, Mengendichte, Subframing / Verfolgung, System-Auslastung, Tests, Zusammenfassung und Ausblick, Konfiguration des Systems, Glossar
ganglion cells, Das Erkennen von Farben im Auge
Gehäuse, Das Auge verglichen mit einer Kamera
Glaskörper, Das Auge verglichen mit einer Kamera
Glossar, Einleitung, Zusammenfassung und Ausblick, Glossar
Grabbing, Der Datenfluss, Übersicht der Implementierten Techniken, Grabbing
Grafikkarte, Aufbau einer Objekterkennung, Konfiguration des Systems, Glossar
Grafisches-Fenster, Der Datenfluss, Übersicht der Implementierten Techniken, Grafisches-Fenster, Xlib-Fenster
Grundlagen, Einleitung, Grundlagen
GUI, Xlib-Fenster, Methoden-Auslastung, Weiterentwicklung und Verbesserung
Güte, Differenzbildung

Stichwortverzeichnis

Hervorhebung, Der Datenfluss
Hintergrund, Grabbing, Segmentierung, Statisches Thresholding, Schwellwertbestimmung mit Hilfe von Histogrammen, Adaptives Thresholding, Differenzbildung, V-Problematik, Kreisflächenähnlichkeit, Algorithmen, Zusammenfassung und Ausblick
 Bereich, Adaptives Thresholding
 Bild, X-Window-Nachrichten
 Farben, Schwellwertbestimmung mit Hilfe von Histogrammen
 Pixel, Adaptives Thresholding, V-Problematik
Histogramm, Schwellwertbestimmung mit Hilfe von Histogrammen, Automatische Schwellwertbestimmung, Adaptives Thresholding
horizontal cells, Das Erkennen von Farben im Auge
Iris, Das Auge verglichen mit einer Kamera
Isodata
 Algorithmus, Automatische Schwellwertbestimmung
Kamera, Farben, Das Auge verglichen mit einer Kamera, Aufbau einer Objekterkennung, Grabbing, Differenzbildung, Subframing / Verfolgung, Grafisches-Fenster, System-Auslastung, Weiterentwicklung und Verbesserung, Zusammenfassung und Ausblick, Konfiguration des Systems, Bedienung der Beispielapplikation, Glossar
 Perspektive, Weiterentwicklung und Verbesserung
 Richtung, Zusammenfassung und Ausblick
 USB, Aufbau einer Objekterkennung, System-Auslastung, Methoden-Auslastung, Zusammenfassung und Ausblick, Konfiguration des Systems, Bedienung der Beispielapplikation
Kaneko, Adaptives Thresholding
Kantenerkennung, Das Erkennen von Farben im Auge, Andere Arten der Segmentierung, Differenzbildung, Zusammenfassung und Ausblick
Kernel
 Convolution, Andere Arten der Segmentierung
 Module, Konfiguration des Systems
 System, Konfiguration des Systems, Bedienung der Beispielapplikation
Koordinaten, Das CIE-Farbsystem, Aufbau einer Objekterkennung, Der Datenfluss, Verarbeitungsrichtung, Subframing / Verfolgung
Koordinatensystem, Das CIE-Farbsystem, Verarbeitungsrichtung, Mengendichte
Kreisflächenähnlichkeit, Übersicht der Implementierten Techniken, Kreisflächenähnlichkeit, Algorithmen, Zusammenfassung und Ausblick
Kriterium, Segmentierung, Mengendichte, Kreisflächenähnlichkeit, Die Selektion
Label, V-Problematik, Objektbewertung
 Bild, Der Datenfluss
 Wert, Der Datenfluss
Labels-Tabelle, Der Datenfluss, Methoden-Auslastung
 Bild, Der Datenfluss
Laplace Transformation, Andere Arten der Segmentierung
Linse, Das Auge verglichen mit einer Kamera
Luminanz, Glossar
Marr, Andere Arten der Segmentierung
Maximum, Andere Arten der Segmentierung
Mengendichte, Übersicht der Implementierten Techniken, Mengendichte, Kreisflächenähnlichkeit, Die Selektion, Zusammenfassung und Ausblick
Methoden-Auslastung, Methoden-Auslastung
Mittelpunktvalenz, Das CIE-Farbsystem
Nachbar, Flächenerkennung, Findung der Nachbarn, Randbehandlung, V-Problematik, Kreisflächenähnlichkeit, Algorithmen

Stichwortverzeichnis

Felder, [Findung der Nachbarn](#), [Randbehandlung](#)
Pixel, [Randbehandlung](#), [Algorithmen](#)
Nachricht, [X-Window-Client-Server-Prinzip](#), [X-Window-Nachrichten](#), [Xlib-Fenster](#)
Netzwerkkarte, [Aufbau einer Objekterkennung](#)
Nulldurchgang, [Andere Arten der Segmentierung](#)
Objektbewertung, [Der Datenfluss](#), [Übersicht der Implementierten Techniken](#), [V-Problematik](#),
[Objektbewertung](#), [Kreisflächenähnlichkeit](#), [Zusammenfassung und Ausblick](#)
Objekterkennung, [Die Beispielapplikation](#), [Zusammenfassung und Ausblick](#)
Objekterkennung/Objektverfolgung, [Objekterkennung/Objektverfolgung](#)
Objekterkennung, [Objekterkennung/Objektverfolgung](#) , [Die Beispielapplikation](#)
Objektiv, [Das Auge verglichen mit einer Kamera](#)
Objektverfolgung, [Einleitung](#), [Objekterkennung/Objektverfolgung](#) , [Die Beispielapplikation](#),
[Grafisches-Fenster](#), [System-Auslastung](#), [Algorithmen](#), [Tests](#), [Zusammenfassung und Ausblick](#)
ORDER-Verknüpft, [Glossar](#)
Ordinate, [Verarbeitungsrichtung](#), [Schwellwertbestimmung mit Hilfe von Histogrammen](#), [Automatische Schwellwertbestimmung](#)
PAL, [Aufbau einer Objekterkennung](#), [Glossar](#)
Prozess, [1. Echtzeitbedingung - Pünktlichkeit](#), [2. Echtzeitbedingung - Auslastung](#), [Aufbau einer Objekterkennung](#), [Grabbing](#), [Subframing / Verfolgung](#), [System-Auslastung](#), [Methoden-Auslastung](#), [Glossar](#)
-zeit, [2. Echtzeitbedingung - Auslastung](#)
Puffer, [Grabbing](#)
Purpurgerade, [Das CIE-Farbsystem](#)
Randbehandlung, [Randbehandlung](#)
Rauschen, [Andere Arten der Segmentierung](#), [Differenzbildung](#), [Beleuchtung](#), [Findung der Nachbarn](#),
[Algorithmen](#), [Glossar](#)
Rechenleistung, [Vorwort](#), [Einleitung](#), [Echtzeit](#), [Grabbing](#), [Adaptives Thresholding](#), [Andere Arten der Segmentierung](#), [Findung der Nachbarn](#), [Mengendichte](#), [Subframing / Verfolgung](#),
[X-Window-Client-Server-Prinzip](#), [System-Auslastung](#), [Methoden-Auslastung](#), [Algorithmen](#),
[Weiterentwicklung und Verbesserung](#), [Zusammenfassung und Ausblick](#)
rekursiv, [V-Problematik](#)
Rennwagen, [Einleitung](#), [Differenzbildung](#), [Mengendichte](#), [Kreisflächenähnlichkeit](#), [Die Selektion](#),
[System-Auslastung](#), [Algorithmen](#), [Weiterentwicklung und Verbesserung](#), [Zusammenfassung und Ausblick](#)
reply, [X-Window-Nachrichten](#)
request, [X-Window-Nachrichten](#)
Retina, [Das Auge verglichen mit einer Kamera](#), [Das Erkennen von Farben im Auge](#)
RGB, [Glossar](#)
 Anteile, [Das CIE-Farbsystem](#)
 Bild, [Die Beispielapplikation](#), [Der Datenfluss](#), [Übersicht der Implementierten Techniken](#),
 [Xlib-Fenster](#), [Methoden-Auslastung](#)
 Farbraum, [Das RGB-Farbsystem](#), [Das CIE-Farbsystem](#)
 Farbsystem, [Das RGB-Farbsystem](#), [Das CIE-Farbsystem](#)
 Format, [Das RGB-Farbsystem](#), [Der Datenfluss](#), [Statisches Thresholding](#), [Methoden-Auslastung](#)
 Mischung, [Das CIE-Farbsystem](#)
 System, [Das RGB-Farbsystem](#)
Roberts Cross, [Andere Arten der Segmentierung](#)
rods, [Das Erkennen von Farben im Auge](#)
Round-trip, [X-Window-Nachrichten](#)
Schliereneffekt, [Differenzbildung](#), [Glossar](#)
Schwellwert, [Statisches Thresholding](#), [Schwellwertbestimmung mit Hilfe von Histogrammen](#), [Automatische Schwellwertbestimmung](#), [Adaptives Thresholding](#)
Schwellwertbestimmung mit Hilfe von Histogrammen, [Schwellwertbestimmung mit Hilfe von Histogrammen](#)

Stichwortverzeichnis

Sclera, Das Auge verglichen mit einer Kamera
Segmentierung, Der Datenfluss, Übersicht der Implementierten Techniken, Segmentierung,
Verarbeitungsrichtung, Andere Arten der Segmentierung, Beleuchtung, Mengendichte, Grafisches-Fenster,
Xlib-Fenster, Weiterentwicklung und Verbesserung, Zusammenfassung und Ausblick
Sehnerv, Das Auge verglichen mit einer Kamera
Selektion, Übersicht der Implementierten Techniken, Objektbewertung, Die Selektion, Subframing /
Verfolgung, Algorithmen
Server, X-Window-Client-Server-Prinzip, X-Window-Nachrichten, Xlib-Fenster
Sobel, Andere Arten der Segmentierung
Spektralfarbenzug, Das CIE-Farbsystem
Statisches Threshold, Übersicht der Implementierten Techniken
Statisches Thresholding, Statisches Thresholding
Störung, Andere Arten der Segmentierung, Subframing / Verfolgung, Glossar
Subframe, Der Datenfluss, Subframing / Verfolgung, Grafisches-Fenster, System-Auslastung,
Methoden-Auslastung, Algorithmen, Weiterentwicklung und Verbesserung
Subframing, Der Datenfluss, Übersicht der Implementierten Techniken, Subframing / Verfolgung
Subframing / Verfolgung, Subframing / Verfolgung
Subtraktive Farbmischung, Das RGB-Farbsystem
System-Auslastung, System-Auslastung
System-Call, Aufbau einer Objekterkennung, Grabbing, System-Auslastung, Glossar
Test, Findung der Nachbarn, System-Auslastung, Methoden-Auslastung, Algorithmen, Tests,
Weiterentwicklung und Verbesserung
 -aufbau, Tests, Konfiguration des Systems
 -bild, V-Problematik, Tests
 -film, Tests
 -lauf, Tests
 -zenario, System-Auslastung
Threshold, Statisches Thresholding, Automatische Schwellwertbestimmung, Adaptives Thresholding, Andere
Arten der Segmentierung, Flächenerkennung, Xlib-Fenster, Weiterentwicklung und Verbesserung
 -berich, Weiterentwicklung und Verbesserung
 -funktion, Flächenerkennung
 -niveau, Statisches Thresholding, Adaptives Thresholding
 -wert, Automatische Schwellwertbestimmung, Adaptives Thresholding, System-Auslastung,
 Methoden-Auslastung, Bedienung der Beispielapplikation
 Filter, Statisches Thresholding, System-Auslastung, Konfiguration des Systems
 Level, Methoden-Auslastung
Thresholding, Der Datenfluss, Übersicht der Implementierten Techniken, Schwellwertbestimmung mit Hilfe
von Histogrammen, Adaptives Thresholding, Flächenerkennung, Findung der Nachbarn, Objektbewertung,
Subframing / Verfolgung, Methoden-Auslastung, Algorithmen, Tests, Zusammenfassung und Ausblick
 -niveau, Differenzbildung
Unbuntpunkt, Das CIE-Farbsystem
UND-Verknüpft, Statisches Thresholding, Glossar
USB, Aufbau einer Objekterkennung, System-Auslastung, Methoden-Auslastung, Zusammenfassung und
Ausblick, Konfiguration des Systems, Bedienung der Beispielapplikation, Glossar
V-Problematik, V-Problematik, Tests
Verarbeitungsrichtung, Verarbeitungsrichtung, Flächenerkennung
Verbesserung, Weiterentwicklung und Verbesserung
Verfolgung, Einleitung, Aufbau einer Objekterkennung, Der Datenfluss, Übersicht der Implementierten
Techniken, Die Selektion, Subframing / Verfolgung, Xlib-Fenster, System-Auslastung,
Methoden-Auslastung, Algorithmen, Weiterentwicklung und Verbesserung, Zusammenfassung und Ausblick,
Bedienung der Beispielapplikation

Stichwortverzeichnis

Verwindung, [Andere Arten der Segmentierung](#)
Video4Linux, [Aufbau einer Objekterkennung](#), [Grabbing](#), [Konfiguration des Systems](#), [Glossar](#)
Videoausgang, [Das Auge verglichen mit einer Kamera](#)
virtuelle Primärvalenzen, [Das CIE-Farbsystem](#)
vordere Augenkammer, [Das Auge verglichen mit einer Kamera](#)
vordere Fokussierungslinse, [Das Auge verglichen mit einer Kamera](#)
Vordergrund, [Segmentierung](#), [Statisches Thresholding](#), [Schwellwertbestimmung mit Hilfe von Histogrammen](#), [Adaptives Thresholding](#)
 -element, [Statisches Thresholding](#)
 Pixel, [Der Datenfluss](#), [Adaptives Thresholding](#), [Flächenerkennung](#), [Findung der Nachbarn](#),
 [Objektbewertung](#), [Mengendichte](#), [Methoden-Auslastung](#)
Vovea, [Das Auge verglichen mit einer Kamera](#)
Weiterentwicklung, [Einleitung](#), [Weiterentwicklung und Verbesserung](#)
X-Window-Nachrichten, [X-Window-Nachrichten](#)
Xlib, [Grafisches-Fenster](#), [X-Window-Nachrichten](#), [Xlib-Fenster](#), [Konfiguration des Systems](#)
 Anweisung, [X-Window-Nachrichten](#)
 Aufruf, [X-Window-Nachrichten](#)
 Bibliothek, [Zusammenfassung und Ausblick](#)
 Fenster, [Xlib-Fenster](#)
 Funktion, [Übersicht der Implementierten Techniken](#), [Grafisches-Fenster](#), [X-Window-Nachrichten](#)
 Programm, [X-Window-Nachrichten](#)
Zonularfasern, [Das Auge verglichen mit einer Kamera](#)
Zusammenfassung und Ausblick, [Zusammenfassung und Ausblick](#)
zweite Echtzeitbedingung, [2. Echtzeitbedingung - Auslastung](#)

[Zurück](#)

[Zum Anfang](#)

Literatur